UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INFORMÁTICA

PROYECTO FIN DE CARRERA

SISTEMA DE CONTROL DE RUTAS

(Aplicación Web y Android)

- *Autor del proyecto: Verónica Pérez Pérez* (I.T. Informática de Gestión)
- *Director del proyecto: Luis Fernando de Mingo López*

- *Fecha: 20-04-2016*

ÍNDICE

1. Introducción

Una aplicación móvil o app (inglés), es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Por lo general se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles como Android, iOS, BlackBerry OS, Windows Phone, entre otros.

Para completar la funcionalidad de la aplicación móvil he creado una aplicación web. Una aplicación web es una aplicación software que se codifica en un lenguaje soportado por los navegadores web. Los usuarios acceden a ellas mediante un servidor web a través de Internet o de una intranet mediante un navegador.

Una vez que nos hemos puesto en el contexto sobre el que se sitúa el proyecto, explicaré para qué está destinado.

La aplicación está destinada a una empresa de logística para controlar las rutas que realizan sus vehículos.

En una empresa de logística la optimización en tiempo y coste es muy importante. Con el control de las rutas se podrán comprobar que ruta han seguido los vehículos y el tiempo destinado a ello.

Una necesidad importante también puede ser comunicarse con el conductor en algún momento de la ruta. Como se puede encontrar conduciendo y no poder atender una llamada, la mejor opción es una notificación al dispositivo móvil. El conductor oirá el sonido de la recepción de una notificación, y en cuanto pueda, parará y podrá leer la notificación tranquilamente.

1.1. Objetivos

El objetivo del proyecto es desarrollar una aplicación de geolocalización con tecnología Android y una aplicación web que interactúe con la aplicación Android, mediante notificaciones Push utilizando el servidor de notificaciones de Android, Google Cloud Messaging.

Con el desarrollo de las dos aplicaciones aprenderé la estructura de una aplicación Android y una aplicación web, en ambos utilizando como principal lenguaje de programación, un lenguaje orientado a objetos, como es Java.

También conoceré como comunicar una aplicación Android con otra web, mediante JSON, JavaScript Object Notation.

Me he decidido por un proyecto de este tipo porque aúna dos tecnologías que está creciendo mucho actualmente. Además me permite poner en práctica mucho de los conocimientos adquiridos a lo largo de la carrera de Ingeniería Técnica de Informática de Gestión.

2. Aplicación Web de gestión

2.1. Tecnología y estructura de un proyecto web

2.1.1. Patrón MVC

El patrón MVC se divide en tres partes:

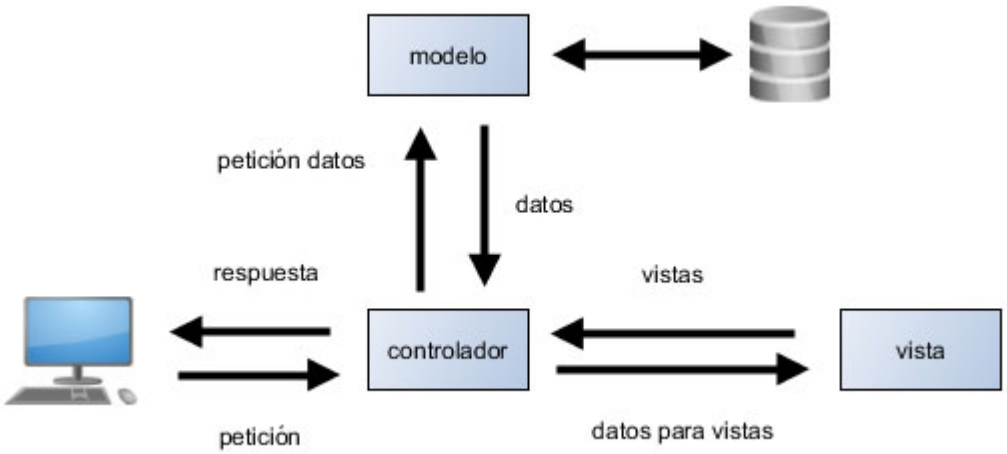
- Un modelo que representa la lógica de los datos de la aplicación
- La vista, que es la representación visual de los datos
- Un controlador, que actúa de intermediario entre el modelo y la vista

Descripción

Modelo: el modelo es la representación lógica de la información y describe la funcionalidad del sistema, por lo tanto se encarga de se encarga de realizar la recuperación, actualización, inserción y eliminación de datos, a través de una lógica de negocio.

Controlador: es intermediario entre la vista y el modelo. Responde a eventos generados por el usuario el cual constituyen llamados al **Modelo** (en el caso que se solicite alguna información como lista de clientes, consultar un proveedor o editar un producto) o a la **Vista** (en el caso de mostrar un formulario o un reporte estadístico). En pocas palabras el componente que hace posible tener separada la lógica de negocio con la vista es el **Controlador**.

Vista: se refiere a toda la interacción con el usuario como los formularios o listados (en este proyecto la vista genera el **HTML** necesario).



Conclusiones

El patrón MVC es muy utilizado hoy en día, es un diseño que ofrece consistencia y baja complejidad en el desarrollo de software y podemos encontrar muchos frameworks que implementan esta solución. En concepto es muy sencillo, una capa intermedia para comunicar los datos o la lógica del negocio con la vista, de este modo si queremos modificar la vista no tenemos que modificar la lógica de negocio y viceversa.

2.1.2. Play framework Java

Play es un framework de aplicaciones web de código abierto, de alta productividad que integra los componentes y APIs necesarias para el desarrollo moderno de aplicaciones web .

Se basa en una arquitectura ligera , sin estado, en un entorno web amigable y cuenta con el consumo de recursos predecible y mínimo (CPU , memoria , hilos) para aplicaciones altamente escalables gracias a su modelo reactivo.

Está escrito en Java y Scala y se puede desarrollar con cualquier de los dos lenguajes. En el proyecto he utilizado el lenguaje Java, por eso a partir de aquí obviaré temas referentes a lenguaje Scala.

Anatomía de una aplicación Play

El diseño de una aplicación de Play ha sido estandarizada para mantener las cosas lo más simple posible

- Directorio /app/

Este directorio contiene todo lo ejecutable: Código fuente Java, plantillas y fuentes compilados.
Además, hay tres paquetes estándar en el directorio de aplicación, uno para cada componente del patrón de arquitectura MVC:

/app/controllers
/app/models
/app/views

Por supuesto, se pueden añadir otros paquetes, por ejemplo, un paquete app/utils.

- Directorio /public/

Los recursos almacenados en el directorio público son activos estáticos que son atendidos directamente por el servidor Web.
Este directorio se divide en tres sub- directorios uno para las imágenes, otro para las hojas de estilo CSS y otro para los archivos JavaScript .

- Directorio /Config/

Contiene los archivos de configuración de la aplicación. Hay dos archivos de configuración principales:

application.conf, el archivo de configuración principal de la aplicación , que contiene los parámetros de configuración estándar
routes, el archivo de rutas definición.

- Directorio /lib/

Este directorio es opcional y contiene las librerías adicionales, las que están fuera del sistema de compilación.

- built.sbt fichero

Fichero que contiene las principales declaraciones para construir el proyecto.

- Directorio /project/

Contiene las definiciones de compilación:

plugins.sbt, define los plugins utilizados en el proyecto
build.properties contiene la versión de compilación utilizada para construir la aplicación

- Directorio /.target/

Contiene todo lo generado por el sistema de compilación (clases compiladas).

2.1.3. HTML

HTML es un lenguaje de marcas para la elaboración de páginas web, **HyperText Markup Language** (lenguaje de marcas de hipertexto). Un lenguaje de marcas es una forma de codificar un documento,

que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.
HTML es un estándar a cargo de la organización W3C, que se dedicada a la estandarización de casi todas las tecnologías ligadas a la web.

Este lenguaje basa su desarrollo en la diferenciación. Los elementos externos a la página (imagen, vídeo, script, etc) no se incrustan directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene sólo texto y es el navegador web (interpretador del código) el que une todos los elementos y visualizar la página final. La idea es que cualquier página web, indistintamente de la versión en la que haya sido desarrollada, pueda ser interpretada por cualquier navegador actualizado.

Los elementos tienen dos propiedades básicas: atributos y contenido. Un elemento, generalmente tiene una etiqueta de inicio y otra de cierre. Y los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas.

2.1.4. CSS

CSS, cascading style sheets (hoja de estilo en cascada), es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML. El W3C es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los navegadores.
Con este lenguaje se separa la estructura de un documento de su presentación, con esto se puede conseguir definir estilos generales o para cada etiqueta particular. También se centraliza el control de la presentación con lo que se agiliza la actualización del mismo. Se optimiza el ancho de banda de la conexión, ya que pueden definirse los mismos estilos para muchos elementos o porque un mismo archivo CSS puede servir para una multitud de documentos.

Este lenguaje tiene una sintaxis muy sencilla. Utiliza unas palabras clave en inglés para especificar los nombres de varias propiedades de estilo. Una hoja de estilo se compone de una lista de reglas. Cada regla o conjunto de reglas consiste en uno o más selectores y un bloque de declaración con los estilos a aplicar para los elementos del documento que cumplan con el selector que les precede. Cada bloque de estilos se define entre llaves, y está formado por una o varias declaraciones de estilo con el formato propiedad:valor.

2.1.5. Javascript

JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web que permite mejoras en la interfaz de usuario y páginas web dinámicas, aunque también existe una forma de JavaScript del lado del servidor.
La sintaxis de JavaScript es similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.
Todos los navegadores interpretan el código JavaScript integrado en las páginas web. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

2.2. Entorno de desarrollo

En este punto se detalla el proceso de instalación de todas las herramientas necesarias para el desarrollo de la aplicación web. Se necesita instalar Java, una base de datos, una API de Google, un IDE y una aplicación de control de versiones de software.

2.2.1. Java

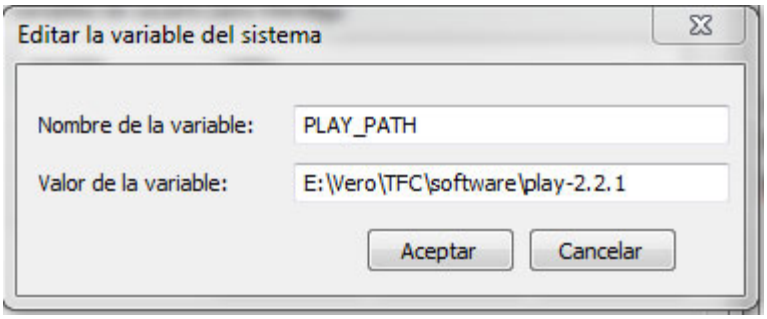
Para poder programar en el framework Play, utilizando el lenguaje de programación Java, se necesita tener instalado Java en el Sistema Operativo. Para la versión de Play utilizada, 2.2.1, se necesita un JDK (Java Development Kit) versión 6 o superior. Para el proyecto he utilizado la versión 7. Para descargarlo, lo mejor es acceder a la página oficial *http://www.oracle.com/technetwork/es/java/javase/downloads/jdk7-downloads-1880260.html* .

```
C:\>java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)
```

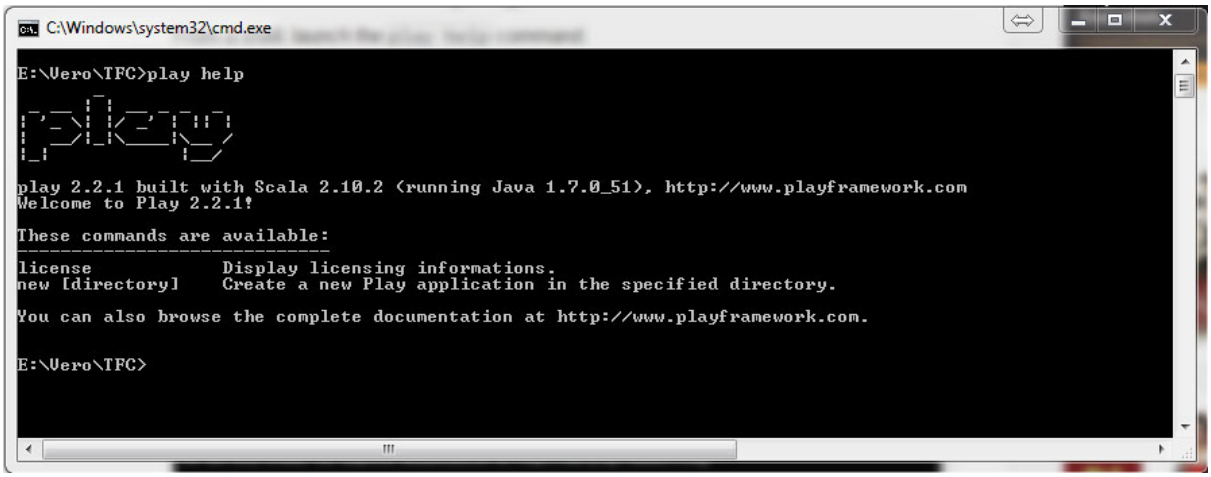
2.2.2. Play

Descargo la versión de Play con la voy a trabajar, en mi caso utilizo la versión 2.2.1, con Scala 2.10.2, porque considero que ya está lo suficientemente probada y estable, además que encuentro más ayuda. Extraigo los ficheros a un directorio donde tengo permisos de lectura y escritura.

Se debe añadir el directorio de instalación del framework al PATH del sistema. En Windows se tiene que crear una variable global de entorno.



Después de estos pasos compruebo que Play está disponible desde la consola de comandos.



2.2.3. H2-Browser

El framework Play viene con una base de datos autónoma llamada H2 Browser. Es un motor de base de datos Open Source desarrollado en Java. No es una base de datos comparable con MySQL, SQL Server u Oracle, pero para comenzar mi proyecto en local es suficiente. Puntualizo local, porque para subir la aplicación web a producción, utilizaré otra base de datos en la nube, una que proporciona Heroku.

Las características más importantes de esta herramienta son:

- Alta integración: Debido a que está implementada en Java, su integración con cualquier aplicación en este lenguaje es total (mediante API JDBC u ODBC).
- Uso en diferentes plataformas: Debido a que es Java se puede utilizar en cualquier plataforma.
- Rápida: Debido en su estrategia de optimización basada en costes.
- Tamaño reducido: Ocupa muchísimo menos que muchas de las base de de datos mencionadas anteriormente (JAR aproximado de 1MB).

- Modo embebido: Permite el funcionamiento es este modo realizando la gestión de los datos en archivos haciendo uso de una pequeña parte de memoria.
- Modo en memoria: Permite el funcionamiento en este modo realizando la gestión de los datos directamente sobre la memoria, lo que acelera las operaciones realizadas.

La configuración de la base de datos se encuentra en el fichero application.conf y por defecto la configuración viene comentada, se tiene que descomentar y dejar los siguientes parámetros:

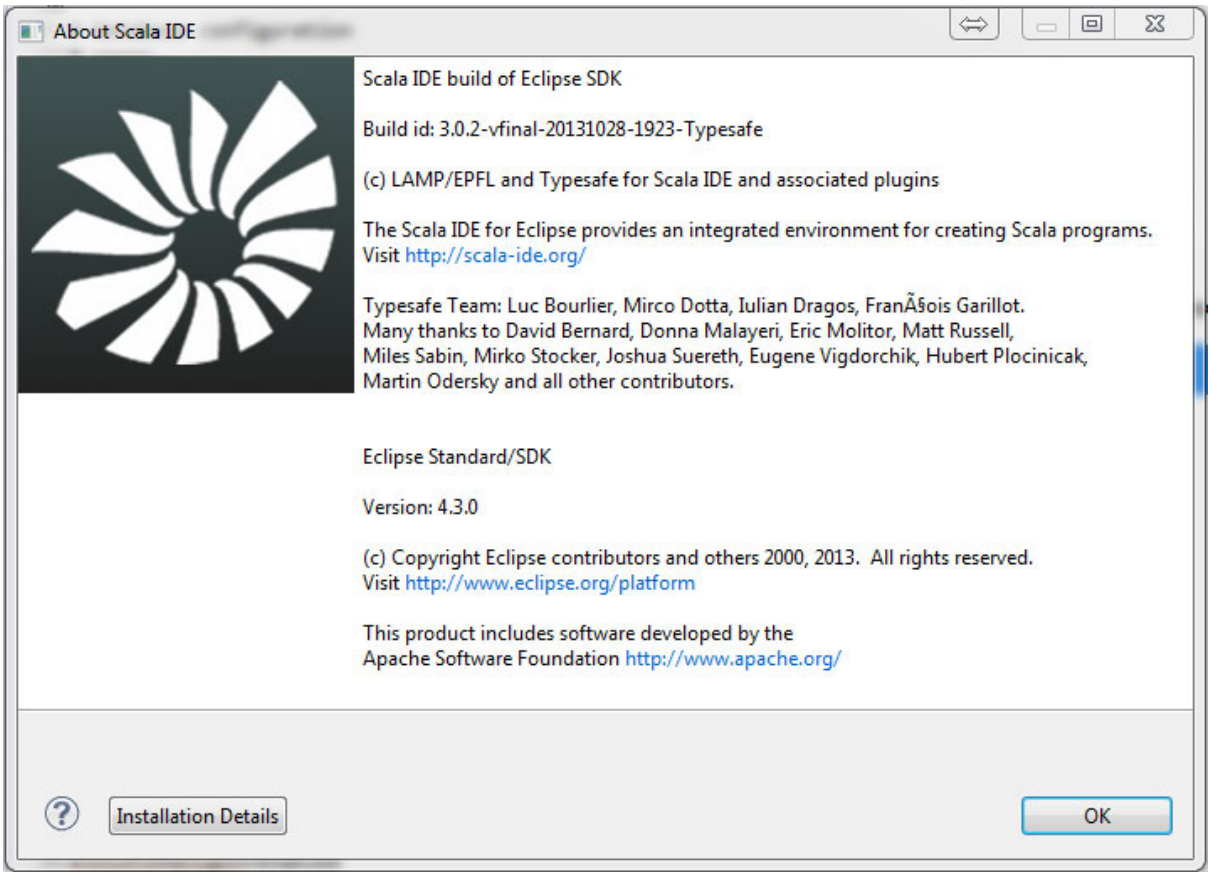
db.default.driver=org.h2.Driver
db.default.url="aqui se pondrá el directorio de la bbdd"

db.default.user=sa
db.default.password=

2.2.4. Eclipse Scala

Para desarrollar la aplicación web necesitamos un IDE(Integrated Development Enviroment). El más utilizado y recomendado para el desarrollo en Play es Eclipse, además es gratuito.

Podemos descargarnos un Scala IDE para Eclipse de http://scala-ide.org/download/sdk.html. La versión que he utilizado para la versión de framework instalado, es la 4.3.0.



2.2.5. Git

Para almacenar y controlar las versiones del código fuente de la aplicación es necesario instalar un sistema de control de código fuente. Existen varios VCS (version control system), yo me he decantado por Git, muy popular actualmente.

Git es un sistema de control de versiones distribuido, de código abierto y gratuito. Lo podemos descargar de http://git-scm.com/download.

Como el proyecto solo lo estoy desarrollando yo sola, no voy a entrar en detalles de configuración de usuarios, comentaré unas pinceladas básicas, ya que no es el fin del proyecto.

Crear un directorio de trabajo de git en la carpeta principal de la aplicación:

```
$ git init
```

Solo crea un directorio de control, no como otros sistemas de control de versiones que generan un directorio de control por cada subdirectorio del proyecto.

Agregar los archivos de la aplicación y confirmarlos:

```
$ git add .
$ git commit -m “init”
```

Modificar ficheros, bien porque se ha cambiado su contenido o porque son nuevos:

```
$ git add file1 file2 file3
```

Aclarar que add no significa añadir al repositorio, los cambios no se versionarán hasta que se haga el commit. Este comando significa añadir al índice para el próximo commit.

Para ver que tenemos pendiente de hacer commit:

```
$ git diff --cached
```

La opción --cached indica que queremos ver los cambios con respecto a lo que ya tenemos añadido al index y que se tendrá en cuenta en el próximo commit. Si no ponemos esta opción, nos indicará todas las cosas que potencialmente podríamos añadir al index con un add.

Otra forma de ver los cambios pendientes es con:

```
$ git status
```

Para crear una nueva versión:

```
$ git commit
```

2.2.6. Heroku

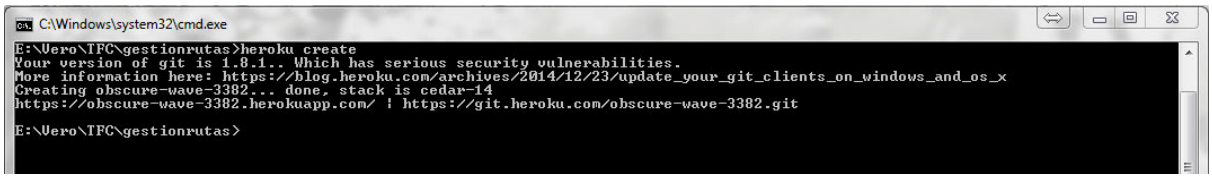
Heroku es una plataforma de aplicaciones en la nube, es una de las principales plataformas para desplegar aplicaciones web. Es gratuito hasta 5MB de espacio en disco para base de datos y 50MB para todos los archivos incluyendo el repositorio Git. El servicio está basado en la nube de Amazon Web Services. El servicio de base de datos que maneja es Heroku Postgres. Éste es accesible desde cualquier lenguaje con un driver Postgres SQL.

Los primeros pasos son:

- Instalarnos la herramienta: https://toolbelt.heroku.com/
- Crear una cuenta en: www.heroku.com

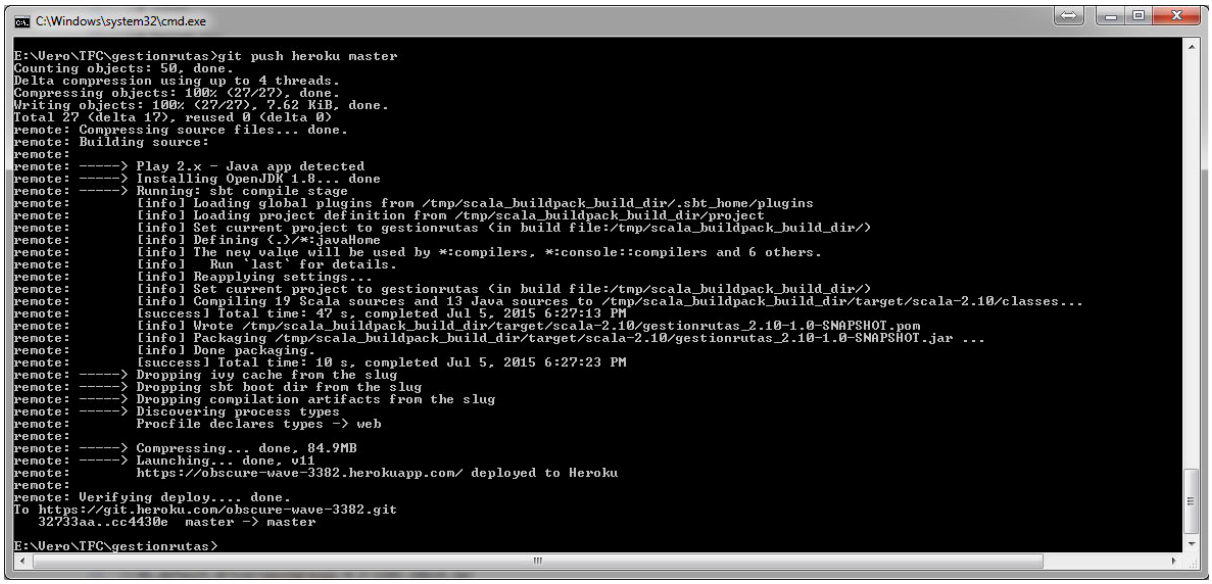
Para crear un proyecto se ejecuta el siguiente comando:

```
$ heroku create
```



Para desplegar la aplicación y subirla al repositorio remoto de Heroku:

$ git push heroku master



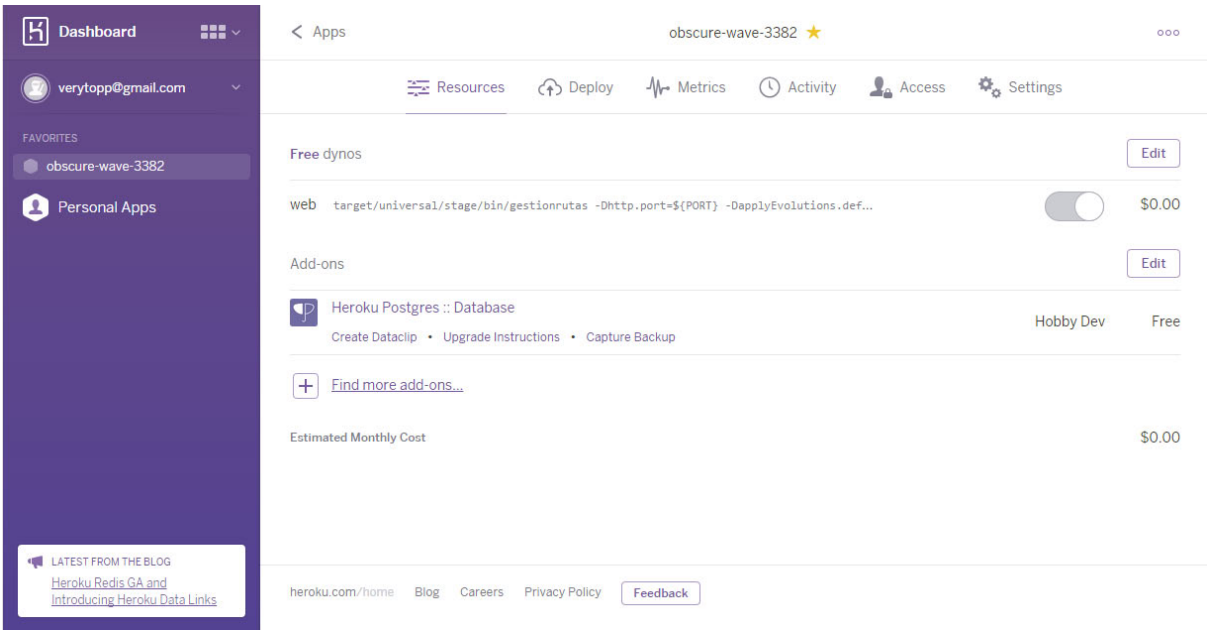
Para comprobar que se la aplicación se ha desplegado correctamente:

$ heroku ps

Para consultar el log en cualquier momento:

$ heroku logs

Después de crear y desplegar la aplicación, en la cuenta de Heroku se puede acceder a la aplicación y a sus datos de configuración.



Como comentaba más arriba Heroku provisiona una BBDD automáticamente. El siguiente paso es configurar la aplicación para usar esa BBDD. Lo primero es añadir a las dependencias de la aplicación el PostgreSQL JDBC driver. Esto consiste en incluir en el fichero build.sbt, la siguiente línea:

libraryDependencies += "postgresql" % "postgresql" % "9.1-901-1.jdbc4"

También es necesario crear, en el directorio raíz, un fichero nuevo que se llame Procfile, y contenga lo siguiente:

web: target/universal/stage/bin/gestionrutas -Dhttp.port=${PORT} -DapplyEvolutions.default=true -Ddb.default.driver=org.postgresql.Driver -Ddb.default.url=${DATABASE_URL}

En este fichero se le dice la variable de entorno donde tiene que ir a buscar la ruta de la base de datos. Esa variable de entorno contendrá la ruta de la BBDD que proporciona Heroku.

Por otro lado, en el fichero de configuración de la aplicación se tiene que meter los datos de acceso, usuario, contraseña y URL, a dicha base de datos, que se obtienen de la consola web.

3. Aplicación Android

3.1. Introducción Android

Android es la primera plataforma de código fuente para aplicaciones móviles con posibilidad de adecuarse a diferentes mercados. Android es un producto de Google, en concreto de la Open Handset Alliance, una alianza formada, actualmente, por 84 organizaciones de tecnología y móviles, que se han unido para desarrollar la primera plataforma móvil completa, abierta y libre.

Android se basa en un núcleo de Linux y en una avanzada máquina virtual optimizada para sus aplicaciones de Java. Ambas tecnologías son esenciales para Android. El núcleo de Linux ofrece agilidad y portabilidad para aprovechar las numerosas opciones de hardware de los futuros teléfonos equipados con Android. El entorno de Java de Android es fundamental porque hace que sea accesible para los programadores debido al número de desarrolladores de software para Java y del completo entorno que ofrece la programación con Java.

3.2. Ventajas

Como decía en el punto anterior, Android es muy accesible para los programadores, pero no solo esa característica es su principal atractivo. A continuación detallo algunas características que hacen que hacen que un desarrollador se decante por esta tecnología.

Plataforma abierta

El sistema operativo Android es una plataforma abierta, así, la comunidad de desarrolladores proporciona los elementos de los que carece. Permite a los fabricantes de hardware desarrollar características específicas para sus dispositivos y así aprovecharlos al máximo. No es necesario depender de Google para disfrutar de nuevas funcionalidades, se pueden obtener librerías de terceros.

Esta característica es la más popular entre consumidores y desarrolladores y ha conseguido un fuerte crecimiento en el uso de aplicaciones.

Abstracción del hardware

Al tener un núcleo de Linux esto proporciona un nivel de abstracción del hardware que permite conservar los niveles superiores independientemente de los cambios del hardware. Las aplicaciones no pueden permitirse fallar en caso de que falte un recurso. Con la aparición de nuevos accesorios en el mercado, se pueden crear controladores en el nivel de Linux para proporcionar asistencia.

Android Market es un mercado libre

Se puede definir el mercado de Android como un mercado libre que ha sido posible por la naturaleza de código abierto de Android. Por ejemplo, cuando se desarrolla una aplicación, el fabricante de la aplicación obtiene el derecho a difundir la aplicación en Internet. Esto ofrece acceso a una enorme audiencia, que hace atractivo desarrollar en esta plataforma.

Estabilidad

El núcleo de Linux es una plataforma demostrada. Esta plataforma proporciona mejor estabilidad y seguridad que otros sistemas operativos móviles. Una plataforma tan completa como Linux proporciona gran potencia y funciones a Android. El uso de una base de código abierto desata la capacidad de individuos y empresas para impulsar la plataforma. Esto es especialmente importante en el mundo de los dispositivos móviles, donde los productos cambian con tanta rapidez y a los que se le exige mucha fiabilidad.

3.3. Historia de Android

Por el año 2003 se comienza a desarrollar el sistema operativo Android, por la empresa Android Inc., cuyos socios eran Andy Rubin, Rich Miner, Nick Sears y Chris White, pero hasta el años 2005 cuando lo compra Google, no comienza a sonar. En noviembre de 2007 se lanzó la Open Handset Alliance, la cual agrupaba a muchos fabricantes de teléfonos móviles, chipsets y Google y se proporcionó la primera versión de Android, una plataforma para dispositivos móviles construida sobre la versión 2.6 de Linux, junto con el SDK para que los programadores empezaran a crear sus aplicaciones para este sistema.

Aunque los inicios fueran un poco lentos, debido a que se lanzó antes el sistema operativo que el primer móvil, rápidamente se ha colocado como el sistema operativo de móviles más vendido del mundo.

Android ha visto numerosas actualizaciones desde su liberación inicial. Estas actualizaciones de sistema operativo arreglan bugs y agregan nuevas funciones. Curiosamente cada actualización del sistema operativo Android se desarrollada bajo un nombre de un elemento relacionado con postres.

Android 1.0 Apple pie

Android 1.0 Apple Pie fue la primera versión de Android comercial lanzada y este lanzamiento se realizó junto al HTC Dream el 23 de septiembre de 2008. El HTC Dream fue el primer móvil con el sistema operativo Android del mercado. Aunque Android y Google pusieron mucho énfasis en el proyecto, a la competencia directa no le preocupo demasiado. En aquel momento el mercado estaba dominado por Symbian, con su nuevo sistema operativo llamado iOS, que se estaba abriendo camino junto con BlackBerry y asomaba tímidamente Windows Mobile.

Como es lógico, ésta fue la versión más novedosa, pues presentaba las bases para el comienzo de un nuevo sistema operativo a nivel comercial. Las características que se presentan en esta versión son las siguientes:

- Notificaciones en un menú desplegable
- La inclusión de accesos directos (widgets) en el escritorio
- Android Market, que ofrece, totalmente gratis, un catálogo completo de aplicaciones
- Integración con aplicaciones de Google; Google Mail, Contacts y Calendar
- Otras funciones relacionadas con productos Google como Navegador, Google Maps, Google Talk y el reproductor de video YouTube

Android 1.1 Banana Bread

Android 1.1 Banana Bread fue una pequeña actualización publicada el 9 de febrero de 2009. Esta actualización estaba orientada exclusivamente a solucionar pequeños errores detectados, en el HTC Dream, único terminal en el momento. Se añadieron unas pocas nuevas características:

- Posibilidad de guardar archivos adjuntos
- Se añadieron detalles y reseñas sobre lugares y negocios en Google Maps

Android 1.5 Cupcake

Con la introducción de Android 1.5, el 30 de Abril de 2009, se empieza a oir del nombre de Cupcake como referencia a la primera actualización importante del sistema operativo de Google y comienza a calar el hecho de que Google va a utilizar nombres de postres para sus versiones de Android y además en orden alfabético

Con esta versión llegan cambios relevantes. Esta versión se basa en el kernel de linux 2.6.27. Los cambios a destacar son:

- Grabación y reproducción de vídeos con camcorder
- Nuevo teclado con predicción textual
- Soporte Bluetooth
- Nuevos accesos directos en el escritorio
- Transiciones de pantalla animadas
- Subida de videos desde el terminal a YouTube y Picassa
- SDK, el cual permite el desarrollo de aplicaciones por parte de terceros

Android 1.6 Donut

Esta actualización se presenta en septiembre de 2009. Fue en realidad una pequeña actualización que incluía un buen número de mejoras en la interfaz. Pero además realizaron cambios en el núcleo del sistema operativo.

- Añaden un soporte para CDMA/EVDO, 802.1x, VPN y text-to-speech
- Mejoras de rendimiento en búsquedas
- Actualizan la búsqueda por voz
- Nuevo diseño de Android Market
- Mejora en el sistema operativo para añadir compatibilidad con diferentes resoluciones de pantalla
- Rediseñan la interfaz de la aplicación de cámara

Android 2.0/2.1 Eclair

Android 2.0 fue la última de las actualizaciones del año 2009, con esta versión dieron un gran salto. Y a comienzos del 2010 lanzaron versión 2.1 junto con el Nexus One. Fue entonces cuando Android se convierte en una amenaza real para iPhone.

- Mejora de la velocidad de hardware
- Soporte de varios tamaños de pantalla y resolución
- Interfaz de usuario renovada
- Navegador de internet con soporte para HTML5
- GPS gratuito a través de Google Maps Navigation
- Compatibilidad con Microsoft Exchange
- Zoom digital en la cámara
- Soporte flash integrado en la cámara
- Soporte para multi-cuentas de usuario en un mismo terminal
- Introducción de la función “Text to Speech”

Android 2.2 Froyo

Se liberó el 20 de mayo de 2010. Esta versión incluyó mejoras muy significativas en el rendimiento y dieron un gran salto en compatibilidad y adaptación a nuevos dispositivos. Algunas de esas mejoras fueron:

- Mejora en el rendimiento del SO. Consiguen una mejora de la velocidad, de la memoria y de la aplicaciones
- Integran JavaScript V8 del Chrome en la aplicación navegador
- Aparece la función de hotspot con WiFi y la de tethering por USB
- Añaden la función de deshabilitación del tráfico de datos del operador
- Soporte para Adobe Flash 10.1

Android 2.3 Gingerbread

Esta versión salió el 6 de diciembre de 2010 de la mano de Samsung, con el Nexus S. Los cambios más relevantes fueron:

- Rediseñan la interfaz de usuario general de Android
- Añaden soporte para las pantallas de mayores dimensiones y resolución WXGA y más grandes.
- Introducen el soporte nativo en Android para llamadas VoIP SIP
- Añaden reproducción de vídeos WebM/VP8 y decodificación de audio AAC
- Añaden el soporte para Near Field Communication
- Introducen las clásicas funciones de cortar, copiar y pegar en el sistema
- Añaden soporte mejorado para desarrollar código nativo
- Añaden soporte nativo para sensores como giroscopios y barómetros
- Introducen el administrador de descargas para archivos grandes
- Mejoran el apartado de administrador de energía, y el control de las apps con el administrador de tareas.
- Se puede utilizar de manera nativa más cámaras

Android 3.0 Honeycomb

Esta versión llegó en febrero de 2011 y sorprendió con su propuesta radical para adaptarse a los tablets. Pero precisamente al ser una versión para tablets tuvo menos repercusión.

Los cambios se pudieron ir viendo en Android 3.0, Android 3.1 y Android 3.2 a lo largo de ese año. Algunas de las mejoras a destacar fueron:

- Escritorio 3D con rediseño de widgets
- Mejora el sistema multitarea
- Añaden múltiples avances en el navegador web preestablecido
- Añaden el soporte a videochat con Talk
- Mejoran los soportes para redes WiFi
- Añaden compatibilidad con periféricos que usan conexión USB
- Se permite la redimensión de aplicaciones creadas para móviles, para su adaptación a las mayores pantallas de las tabletas.

Android 4.0 Ice Cream Sandwich

La presentan en octubre de 2011. Esta versión alcanza el status de sistema operativo moderno, capaz de empezar a dominar el mercado móvil. Llega la unificación entre tabletas y smartphones.

Algunas de las novedades más importantes que incluyeron fueron:

- Unifican el uso de Android en cualquier dispositivo
- Interfaz Holo
- Se añade la posibilidad de usar botones virtuales en lugar de los táctiles capacitivos anteriores
- Añaden la aceleración por hardware
- Multitarea mejorada. Dan la posibilidad de terminar tareas sacándolas de la lista
- Se añade gestor de consumo de datos de tráfico móvil dando el control de éste al usuario
- Los widgets pasan a una lista similar a la del menú principal
- Mejoras en las notificaciones con descartes y visualización en la pantalla de bloqueo
- Añaden teclas de acceso rápido para hacer capturas de pantalla
- Se añade la función Android Beam para compartir contenidos
- Reconocimiento de voz del usuario
- Se pueden eliminar las aplicaciones personalizadas por la operadora del usuario
- Soporte nativo MKV
- Soporte nativo para lápices táctiles

Android 4.1.x/4.2.x/4.3 Jelly Bean

La actualización llega en 2012, se estrenó en el evento de desarrolladores de Google, en el Google I/O 2012. A día de hoy sigue siendo una de las versiones más extendidas, luego veremos esto con más detalle.

- Mejoras de rendimiento del sistema y de los gráficos con Project Butter
- Mejora la entrada de datos táctiles
- Aparece Google Now, el servicio-asistente de voz inteligente de Google
- Se añade el navegador propio Google Chrome
- Búsqueda mediante voz mejorada
- Las notificaciones interactivas de escritorio añaden novedades de diseño
- Mejoras relacionadas con los ajustes de tamaño de los widgets
- Se mejora la corrección ortográfica y la predicción en el teclado
- Dictado de voz offline
- Se añaden mejoras de rendimiento
- Aparecen nuevas animaciones
- Nuevo panel de control
- Soporte a Miracast con streaming de vídeo y audio
- Rediseño del reloj y widgets relacionados
- Perfiles restringidos para tablets

- Compatible con Bluetooth Smarth
- Autocompletar con teclado
- Mejora del sistema para árabe y hebreo
- Se introduce el Gestual Mode para personas invidentes

Con estas versiones Google se lanzó a la fabricación de tabletas de la mano de fabricantes ya experimentados.

Android 4.4 KitKat

En septiembre de 2013 Google presenta la esperada versión 4.4 KitKat, sorprendiendo por colaborar con Nestlé. Con esta versión piensan en unificar versiones de Android y que desde ese punto todos los dispositivos que salgan al mercado vayan con Android 4.4. Las mejoras son:

- Reducción del consumo de batería
- Reducción de requisitos hardware para funcionar
- Compatibilidad con el perfil MAP de Bluetooth
- Compatibilidad con Chromecast
- Vista web de Chrome
- Subtítulos ocultos
- Administración de dispositivos integrada
- Cambia de pantalla de inicio fácilmente
- Fotografía HDR+
- Conexión por infrarrojos
- Ubicación en Ajustes rápidos
- Más seguridad en las zonas de pruebas de aplicaciones
- Podómetro integrado
- Nueva arquitectura abierta para pagos NFC que funciona con cualquier operador móvil
- Pantalla táctil mejorada

Android 5.0 Lollipop

Android 5.0 se presentó en junio de 2014, aunque no fue hasta octubre cuando estuvo disponible en los nuevos terminales que presentó Google, el Nexus 6 y Nexus 9. Y a partir de diciembre se comienzan a actualizar los dispositivos de otra marca.

Esta nueva versión se enfoca en el número de nuevas plataformas y tecnologías orientadas a Android, como son Android TV, Android Auto, Android Wear y una plataforma para el seguimiento de la salud, Google Fit. Otro detalle importante es el lenguaje de diseño inter-plataformado denominado “Material Desing”

Algunas de las mejoras a destacar son:

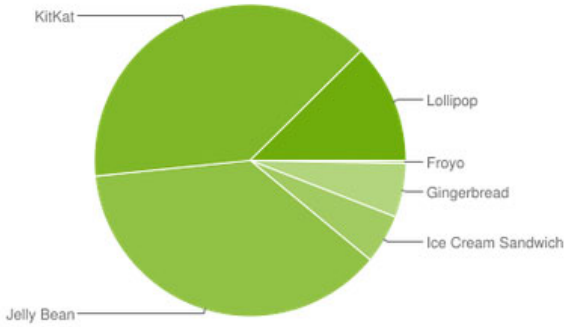
- Nueva interfaz de usuario con un atractivo y colorido diseño que permite disfrutar de una experiencia interactiva, homogénea e intuitiva en todos los dispositivos
- Nuevas formas de controlar cuándo y cómo recibes mensajes
- Mejoras en la gestión y control de la batería
- Mejoras de seguridad. Implementación de SELinux en todas la aplicaciones
- Smart Lock permite proteger el teléfono o tablet vinculandolo a otro dispositivo de confianza
- Flexibilidad para compartir dispositivos, mediante inicio de sesión
- Nuevos ajustes rápidos
- Mejor conexión a Internet en cualquier lugar y funciones de Bluetooth de baja energía más potentes
- Tiempo de ejecución y rendimiento más potente, fluido y rápido
- Gráficos más atractivos y mejores funciones de audio, vídeo y cámara
- Funciones mejoradas para usuarios con problemas de visión y daltónicos

- 15 nuevos idiomas

Después de este repaso a la evolución de Android, se puede decir que el desarrollo ha sido espectacular y que se está convirtiendo en un sistema operativo muy presente en nuestro día a día.
Su cuota de mercado es la más alta del momento, pero volviendo al tema de las versiones, merece la pena mencionar unos datos que dispone Android en su web, muy útiles para desarrolladores, donde muestra el número de dispositivos que comparten una determinada característica, como es la versión de Android o el tamaño de la pantalla. Esta información puede ayudar a seleccionar la versión a partir de la cual queremos que nuestros desarrollos sean compatibles.

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

Data collected during a 7-day period ending on June 1, 2015.
Any versions with less than 0.1% distribution are not shown.



3.4. Estructura de un proyecto Android

Para comenzar una aplicación Android, lo primero es conocer y comprender la estructura general de un proyecto Android.

Cuando se crea un proyecto Android en Android Studio se genera automáticamente la estructura de carpetas necesaria para poder generar la aplicación. Esta estructura será común a cualquier aplicación, es necesaria para que las herramientas del SDK puedan compilar y empaquetar la aplicación correctamente. La estructura de carpetas se agrupan en módulos. Un módulo es el primer nivel de contención dentro de un proyecto, en él se encapsulan tipos específicos de archivos y recursos de código fuente. Hay varios tipos de módulos en un proyecto:

Módulos de aplicaciones Android

Estos módulos contienen el código fuente de la aplicación (src), los archivos de recursos (res), y los ajustes de nivel de aplicación, como el archivo de generación de nivel de módulo, los archivos de recursos, y el archivo Manifest de Android. El contenido de estos módulos es eventualmente incorporado en el archivo .apk que se instala en un dispositivo. Los siguientes directorios y archivos comprenden un módulo de aplicación para Android:

Directorio /src/
Contiene todo el código fuente de la aplicación, el código que gestiona la interfaz gráfica, clases auxiliares, etc. Inicialmente, Android Studio creará por nosotros el código básico de la pantalla principal de la aplicación. En resumen, bajo este directorio estará todo el código Java propio de la aplicación.

Directorio main/res/
Contiene todos los ficheros de recursos necesarios para el proyecto, como imágenes, archivos de diseño y cadenas de texto. Los diferentes tipos de recursos se distribuyen entre las siguientes carpetas.

- /res/anim/. Contiene la definición de las animaciones utilizadas.
- /res/color/. Contiene la definición de colores
- /res/drawable/. Contiene los archivos de mapa de bits (PNG, JPEG o GIF), archivos de imagen 9-Patch, y archivos XML que describen formas Drawable u objetos Drawable que contienen varios estados (normal, *pressed, focused,* etc).
- /res/mipmap/. Contiene los iconos del lanzador de aplicaciones. El sistema Android conserva los recursos en esta carpeta (y carpetas densidad específica como mipmap-xxxhdpi) independientemente de la resolución de la pantalla del dispositivo en el que está instalada la

aplicación. Este comportamiento permite aplicaciones de lanzadores para escoger el icono de mejor resolución de la aplicación para que aparezca en la pantalla principal.

- /res/layout/. Contiene archivos XML que definen diseños de pantalla (o parte de una pantalla).
- /res/menu/. Contiene archivos XML que definen los menús de la aplicación.
- /res/raw/. Contiene recursos adicionales, normalmente en formato distinto a XML. Deben ser referenciados desde la aplicación utilizando un identificador de recursos en la clase R. Por ejemplo, este es un buen lugar para los medios de comunicación, tales como archivos MP3 o Ogg.
- /res/values/. Contiene otros recursos de la aplicación, como por ejemplo, cadenas de texto, estilos y colores. Para los archivos XML que definen los recursos por tipo de elemento XML. A diferencia de otros recursos en el directorio res/, recursos escritos en archivos XML en esta carpeta no se hace referencia por el nombre del archivo. En cambio, el tipo de elemento XML controla cómo los recursos definidos en los archivos XML se colocan en la clase R.
- /res/xml/. Contiene ficheros XML que configuran diversos componentes de la aplicación.
  Por ejemplo, un archivo XML que define un PREFERENCE, AppWidgetProviderInfo, o la capacidad de búsqueda de metadatos.

Directorio /build/

Contiene una serie de ficheros de código Java, generados automáticamente al compilar el proyecto. El más importante es el fichero R.java y la clase R. Esta clase R contendrá en todo momento una serie de constantes con los ID de todos los recursos de la aplicación incluidos en la carpeta /res/, de forma que se podrá acceder fácilmente a estos recursos desde nuestro código a través de este datos.
En Android Studio los ficheros generados se ubican dentro del directorio build de cada módulo. Aún así el acceso a los recursos mediante la clase R es transparente pues el entorno hace la importación automática de la clase cuando la necesitemos.

Directorio /assets/

Contiene otros ficheros auxiliares necesarios (ficheros de datos, fuentes,…). A diferencia de la carpeta /res/raw/, a estos no se les asocia un id en la clase R y se accede ellos por su ruta como cualquier otro fichero del sistema.

Fichero AndroidManifest.xml

Se trata del fichero más importante de una aplicación Android. Es un fichero de control que describe la naturaleza de la aplicación y cada uno de sus componentes. Por ejemplo, describe las actividades y ciertas cualidades sobre ellas, servicios, receptores de intención, y proveedores de contenido; qué permisos se solicitan; qué características de dispositivo se requiere, que niveles de API son compatibles o necesarios; y otros.

Fichero build.gradle

Se trata de los ficheros más importantes de una aplicación Android después del AndroidManifest.xml si trabajamos con el IDE Android Studio, pues muchas de las características de la aplicación se definen recientemente en el fichero de configuración build.gradle, como por ejemplo, los niveles de API.
Usando gradle también podremos configurar distintos perfiles de generación de nuestro APK, por ejemplo para diferentes entornos (desarrollo, preproducción, producción, etc).

Módulos de prueba

Estos módulos contienen código para probar los proyectos y están incorporadas en las aplicaciones de prueba que se ejecutan en un dispositivo. Por defecto, Android Studio crea el módulo androidTest para insertar pruebas JUnit.

Módulos de Biblioteca

Estos módulos contienen código fuente de Android y recursos a los que se puede hacer referencia en proyectos de Android. Esto es útil cuando se tiene código común que se desea volver a utilizar. Los módulos de biblioteca no se pueden instalar en un dispositivo, sin embargo, se incluye en el archivo .apk en tiempo de compilación.

App Engine Módulos

Android Studio permite añadir fácilmente un backend en la nube para una aplicación. Un backend permite implementar funciones como copia de seguridad de los datos del usuario a la nube, sirve contenido a aplicaciones cliente, interacciones en tiempo real, el envío de las notificaciones push a través de Google Cloud Messaging (GCM) para Android, entre otras cosas.

Android Studio provee varios tipos de módulos App Engine:

- App Engine Java Servlet para construir peticiones y enviar datos utilizando HttpClient.
- App Engine Java Endpoints permite a una aplicación cliente de Android hacer llamadas directas a la API del backend.
- App Engine Backend with Google Cloud Messaging para enviar notificaciones push desde el servidor a los dispositivos Android.

3.5. Entorno de desarrollo

3.5.1. Java

Para desarrollar en Android necesitamos tener instalado el Java Development Kit en su versión 7 al menos.
Gracias a desarrollarse en Java, el desarrollo de aplicaciones para Android tiene a su disposición la infinidad de librerías Java disponibles además de todas las nuevas que se van desarrollando enfocadas a su uso en dispositivos Android.

3.5.2. Android Studio

Android Studio es el IDE oficial para el desarrollo de aplicaciones Android. Está basado en el IDE IntelliJ IDEA. Las características más importantes que tiene son:

- Sistema de generación de proyectos basado en Gradle.
- Capacidad para generar distintas variantes de un mismo APK.
- Plantillas de código para ayudar en tareas habituales.
- Editor visual de interfaces gráficas (layout) mejorado.
- Inspección de código basada en *lint* para encontrar problemas de rendimiento o errores comunes.
- Capacidad para firmar aplicaciones y uso de la herramienta ProGuard.
- Soporte integrado de Google Cloud Platform para hacer más fácil la integración de Google Cloud Messaging y App Engine.

3.5.3. API Google

El API de Google que tenemos a nuestra disposición en Android nos permitirá acceder a la gran cantidad de servicios que ofrece Google. Por ejemplo desde Android podremos acceder al API de Google Drive, Google Maps, Google Cloud Messaging, Google Play Games, Google Fit, etc.
Tendremos que configurar algunos de los servicios de Google que vayamos a utilizar vía la web de Google API Developers Console, como para el caso de GCM.

3.5.4. Librerías externas

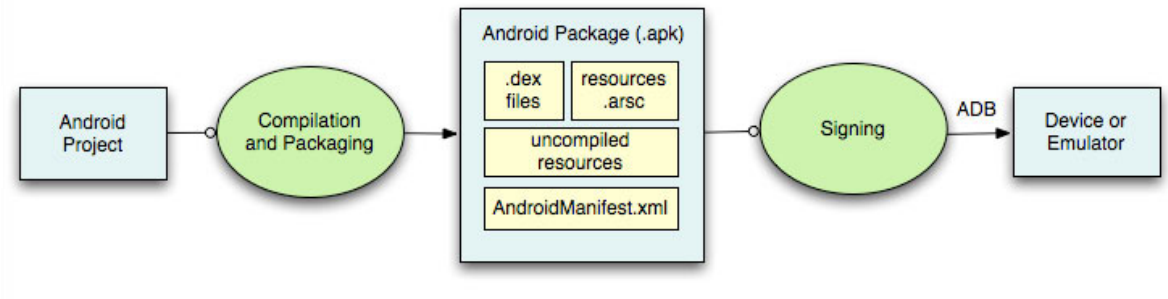
En el desarrollo de cualquier aplicación Android tenemos a nuestra disposición librerías externas, que añaden funcionalidades que nativamente están ausentes o simplemente mejoran el cómo se ejecutan algunas funcionalidades que originalmente son más complejas o menos claras.

3.6. Sistema de generación de un APK

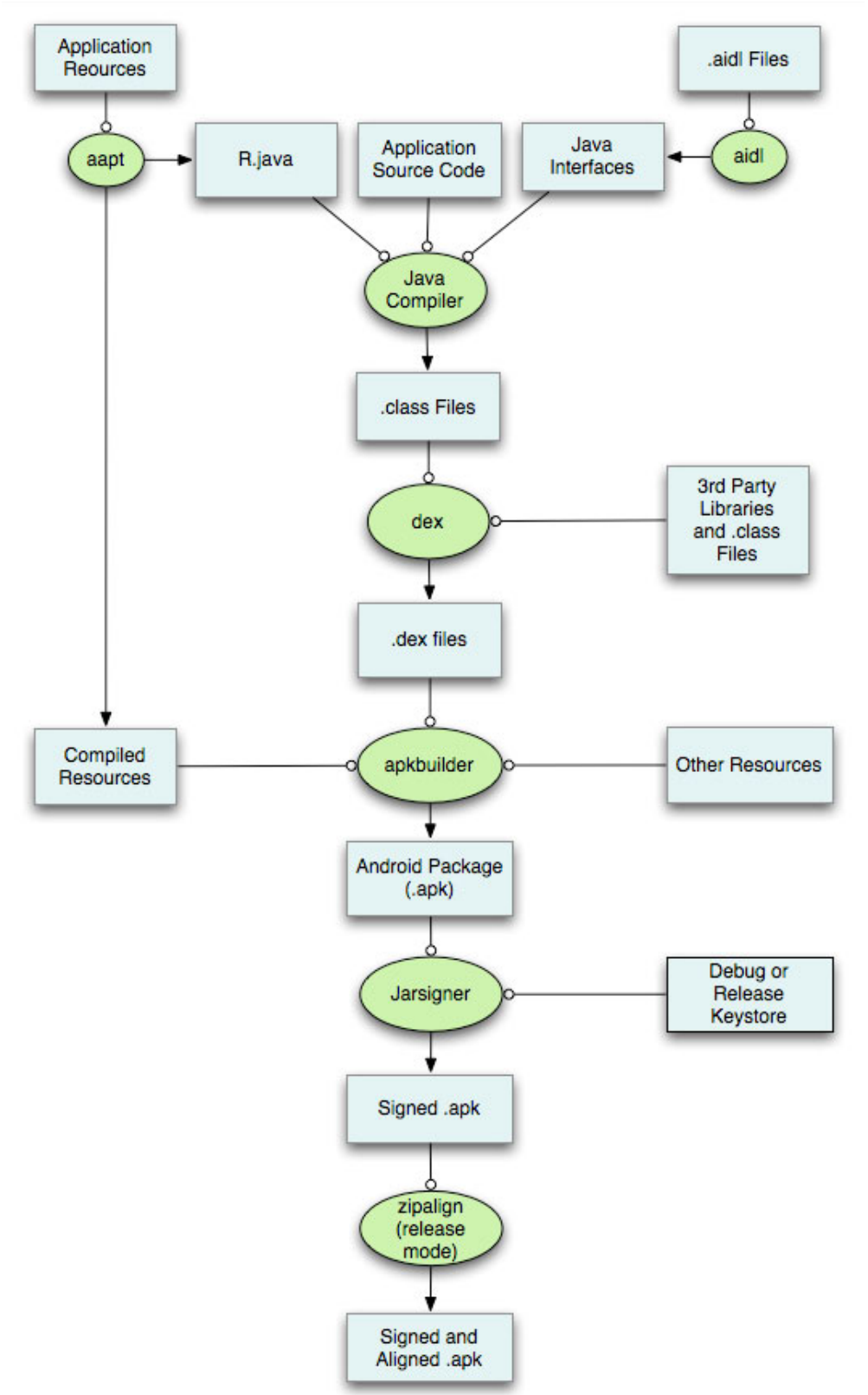
El proceso de construcción implica muchas herramientas y procesos que generan archivos intermedios en la producción del empaquetado en Android (APK). En Android Studio, el proceso de construcción completa se realiza cada vez que se ejecuta el Gradle. El proceso de construcción es muy flexible por lo que es útil entender lo que está sucediendo en ese empaquetado, ya que gran parte del proceso de construcción es configurable y extensible.

Un APK es un fichero empaquetado que incluye el *bytecode* de la aplicación preparado para que sea legible por la máquina virtual del sistema operativo, así como ficheros adicionales de recursos y configuración.

El sistema de generación se resume con este gráfico:



Y un poco más en detalle en este otro diagrama, donde se muestran las diferentes herramientas y procesos que intervienen en la construcción:


Application Reources
.aidl Files
aapt
R.java
Application Source Code
Java Interfaces
aidl
Java Compiler
.class Files
dex
3rd Party Libraries and .class Files
.dex files
Compiled Resources
apkbuilder
Other Resources
Android Package (.apk)
Jarsigner
Debug or Release Keystore
Signed .apk
zipalign (release mode)
Signed and Aligned .apk

A continuación describo los detalles de los pasos del diagrama anterior:

- La herramienta de Android Asset Packaging Tool (aapt) coge los recursos de la aplicación, como por ejemplo el AndroidManifest.xml y los ficheros XML, contenidos en el directorio /res, los compila y se genera la clase R.java.

- El aidl tool convierte cualquier interfaz de tipo Android Interface Definition Language en una interfaz java.

- El compilador de Java, coge la clase R.java, los ficheros generados por el aidl y el código fuente, contenido en el directorio /src, los compila y genera los ficheros .class

- La herramienta dex convierte los ficheros .class a Dalvik byte code. Y junto con algunas librerías externas, todo se incluye y convierte en ficheros .dex, que son los que pueden empaquetar.

- La herramienta apkbuilder, coge todos los recursos no compilados, los compilados y los ficheros .dex y los empaqueta en el fichero .apk

- Una vez, se ha generado el fichero .apk, este debe ser firmado con el debug o release key antes de que la aplicación se instale en un dispositivo.

- Finalmente, si la aplicación ha sido firmada en modo release, el fichero .apk debe ser alineado con la herramienta zipaling. Esto optimiza de manera importante la aplicación Android, decrementando el uso de memoria durante la ejecución en un dispositivo.

3.7. Interfaz de usuario

La interfaz de usuario (UI) de una aplicación Android se crea con un fichero XML que contendrá elementos propios del sistema operativo que se necesitan para mostrar la información al usuario (que serán las *Views*), así como contenedores que podrán agrupar y organizar todos esos elementos en la interfaz de la manera más usable posible (y serán los *ViewGroups*).

3.7.1. Diseño por declaración: XML

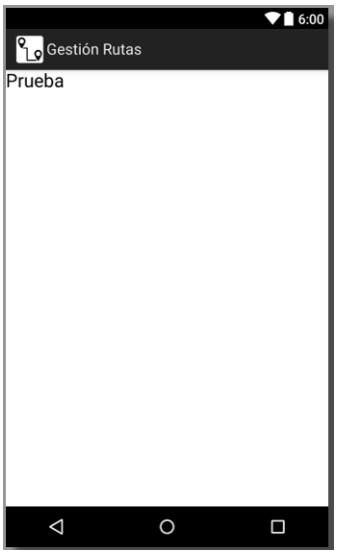
La interfaz de una aplicación se puede desarrollar programáticamente o usando ficheros XML en los que se dispondrán los elementos de la manera que se necesiten. Lo habitual es usar los ficheros XML por su facilidad a la hora de diseñar la interfaz así como por ser más sencillo hacer cualquier cambio o ajuste una vez avanzado el desarrollo de la aplicación. Cada fichero XML definirá un *layout* de la aplicación que podrá contener varios *Views* o *ViewGroups*.

Android Studio dispone además de la posibilidad de editar los ficheros XML de los layouts de manera visual haciendo uso de su vista *Design*. Con esta ayuda se puede, manera sencilla, añadir elementos a nuestro *layout* simplemente arrastrando a la ubicación que deseemos. Desde el mismo editor en modo diseño, se pueden ajustar las propiedades de los elementos, así como organizarlos dentro de contenedores (*ViewGroups*).

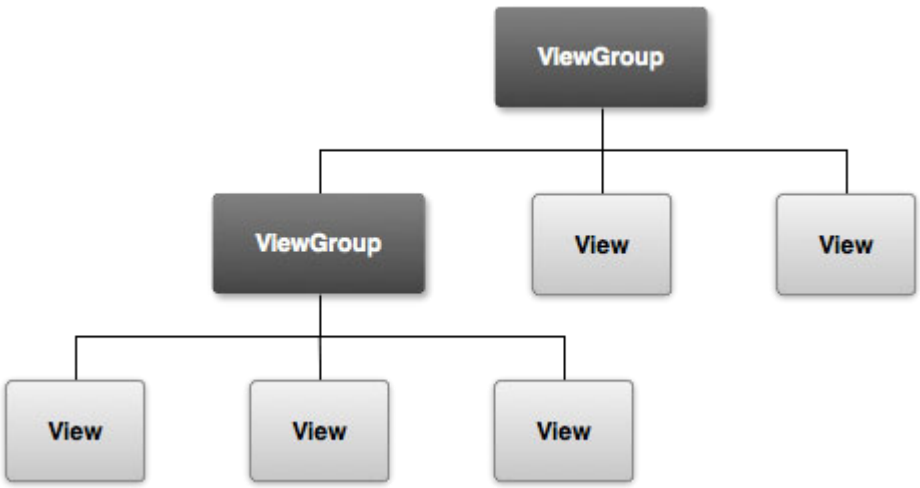
A continuación muestro un ejemplo de un layout básico haciendo uso además de un ViewGroup y un View, los cuales detallaré más adelante:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Prueba"
        android:id="@+id/textView5" />
</LinearLayout>
```

Este layout quedaría en ejecución (o en la vista diseño de Android Studio) de la siguiente manera:



La jerarquía de los elementos que componen una interfaz de usuario se puede ver en la siguiente imagen, las *Views* o widgets están organizadas dentro de los *ViewGroups* o contenedores:



3.7.2. Views

Las *Views (o widgets)* son, como introduje anteriormente, los elementos que contendrá una vista y con los que interactúa o ve el usuario. Estos se organizan dentro de los *ViewGroups* que veremos más adelante.

Las Views se pueden crear, como veíamos antes, programáticamente en Java, o mediante su declaración en el fichero XML que define un layout. A continuación voy a listar los más comunes junto a un ejemplo en XML:

- **TextView**: Sirve para mostrar un texto por pantalla sin poder interactuar el usuario con él, aunque esto puede variar. En el caso del TextView, la propiedad *text* es la más característica pues es en la que se le indica el texto a mostrar. Ejemplo:

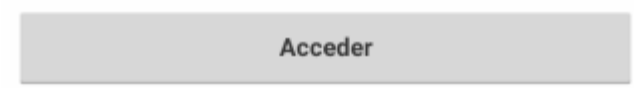
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="NombreUsuario"
    android:id="@+id/usuario_text" />
```

- **EditText**: Sirve para que el usuario introduzca un texto que la aplicación le solicita, como por ejemplo el usuario o la contraseña. La propiedad *hint* es importante para indicar al usuario qué debe escribir en ese recuadro. Ejemplo:

```
<EditText
    android:id="@+id/clave"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Clave"
    android:inputType="textPassword"
    android:maxLines="1"
    android:singleLine="true" />
```

- **Button**: Sirve para crear un botón que realice alguna acción definida en la aplicación cuando sea pulsado. Esta funcionalidad se definirá en el código Java. La propiedad *text* indicará qué texto tendrá el botón. Ejemplo:

```
<Button
    android:id="@+id/login_button"
    style="?android:textAppearanceSmall"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="Acceder"
    android:textStyle="bold" />
```



- **RadioButton**: Sirve para crear un botón de tipo radio, con estado marcado y desmarcado. La funcionalidad de este *widget* se complementa al incluir todos los *RadioButton* que funcionan de manera excluyente dentro de un *ViewGroup* del tipo *RadioGroup* que veremos en el siguiente punto. La propiedad *checked* es la más importante, es la que indica si está seleccionado o no. Es muy útil tanto para representar el clásico botón como para elementos más complejos como botones excluyentes..Ejemplo:

```
<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Prueba"
    android:checked="false"
    android:id="@+id/radioButton" />
```

- **CheckBox**: Sirve para crear un botón de tipo check, con estado marcado y desmarcado. Estos botones están asociados a selecciones múltiples. Como en el caso de los RadioButton su propiedad más importante es checked. Ejemplo:

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Prueba"
    android:checked="false"
    android:id="@+id/checkBox" />
```

- **Switch**: Sirve para crear un botón de dos estados más visual que los checkbox. Se introdujo en el API 14 (Android 4.0) y aporta un estilo más moderno como el de un interruptor. Igualmente la propiedad más importante es checked. Ejemplo:

```
<Switch
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Prueba"
    android:id="@+id/switch1"
    android:checked="true" />
```

- **ImageView**: Sirve para crear una imagen en el layout. La propiedad más importante es *src* que es la que indica qué imagen de las disponibles en nuestros *drawables* será el origen. Ejemplo:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/ic_launcher" />
```

- **ProgressBar**: Sirve para crear un indicador de progreso que se suele utilizar para indicar que la aplicación está realizando una operación. Se suelen ocultar y mostrar según sea necesario programáticamente. Ejemplo:

```
<ProgressBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/progressBar2"
    android:layout_gravity="center_horizontal" />
```


- **Spinner**: Sirve para crear el típico desplegable con una lista de opciones para que el usuario seleccione una. Ejemplo:

```
<Spinner
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/valores_spinner"
    android:id="@+id/spinner" />
```

Los valores que aparecen en el Spinner se pueden introducir en un fichero XML a parte de esta forma, y luego hacer referencia usando la propiedad *entries*:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="valores_spinner">
        <item>Prueba 1</item>
        <item>Prueba 2</item>
        <item>Prueba 3</item>
    </string-array>
</resources>
```

- **SeekBar**: Sirve para crear un control que nos permite seleccionar arrastrando un botón un nivel de progreso. Se suele utilizar en los reproductores de música y vídeo para indicar en qué punto de la reproducción estamos y poder avanzar o retroceder al punto que deseemos. Las propiedades más importantes son *max* y *progress*. Ejemplo:

```
<SeekBar
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/seekBar"
    android:progress="50"
    android:max="100" />
```

- **WebView**: Sirve para incrustar en el *layout* un navegador que cargará la url que le indiquemos programáticamente, ya sea local o remota. Ejemplo:

```
<WebView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/webView" />
```

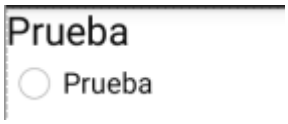
3.7.3. ViewGroups

Los *ViewGroups* son los contenedores que se encargan de organizar las *Views* en un layout y no tienen representación visual. A continuación, como hice con las *Views*, listaré algunos de los más utilizados junto a un ejemplo:

- **LinearLayout**: Sirve para organizar los widgets de manera uno tras otro, ya sea vertical u horizontalmente, haciendo uso de la propiedad *orientation*.

Ejemplo de orientación vertical:

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Prueba"
        android:id="@+id/textView5" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Prueba"
        android:checked="false"
        android:id="@+id/radioButton" />
</LinearLayout>
```



Ejemplo de orientación horizontal:

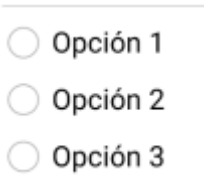
```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
```

- **RelativeLayout**: Sirve para organizar los widgets de manera relativa, es decir, se organizarán siempre relacionados con la ubicación de otro mediante distintas propiedades disponibles. Si no se definen estas propiedades correctamente podrán aparecer unos widgets sobre otros. Ejemplo:

```
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:src="@drawable/ic_launcher" />

    <ProgressBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/progressBar2"
        android:layout_gravity="center_horizontal"
        android:layout_below="@+id/imageView"
        android:layout_toRightOf="@+id/imageView" />
</RelativeLayout>
```

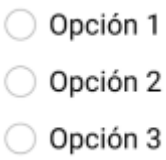
- **RadioGroup**: Sirve para agrupar elementos *RadioButton* y que tengan la funcionalidad habitual de este tipo de elemento de selección exclusiva. El *RadioGroup* hará la gestión de desactivar el resto de *RadioButton* que estén dentro del mismo *RadioGroup* cuando se seleccione uno de ellos. La propiedad más importante es *checkedButton*, que nos dirá cuál de los *RadioButton* está seleccionado. Ejemplo:

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1"
        android:id="@+id/radioButton2" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 2"
        android:id="@+id/radioButton3" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 3"
        android:id="@+id/radioButton4" />
</RadioGroup>
```



- **ScrollView**: Sirve para hacer que la vista incluida dentro de este contenedor tenga un desplazamiento vertical para poder visualizarla completamente si no cabe en la pantalla del dispositivo. Tiene como requisito que sólo puede tener una vista hija. Esa vista hija ya puede tener todas las necesarias que serán las que se aprovechen del desplazamiento vertical para poder visualizarlas si no caben. Como vemos en la segunda captura del ejemplo, no tenemos representación visual como el resto de *ViewGroups*, salvo la del desplazamiento vertical. Ejemplo:

```
<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/scrollView" >

    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Opción 1"
            android:id="@+id/radioButton2" />

        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Opción 2"
            android:id="@+id/radioButton3" />

        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Opción 3"
            android:id="@+id/radioButton4" />
    </RadioGroup>
</ScrollView>
```

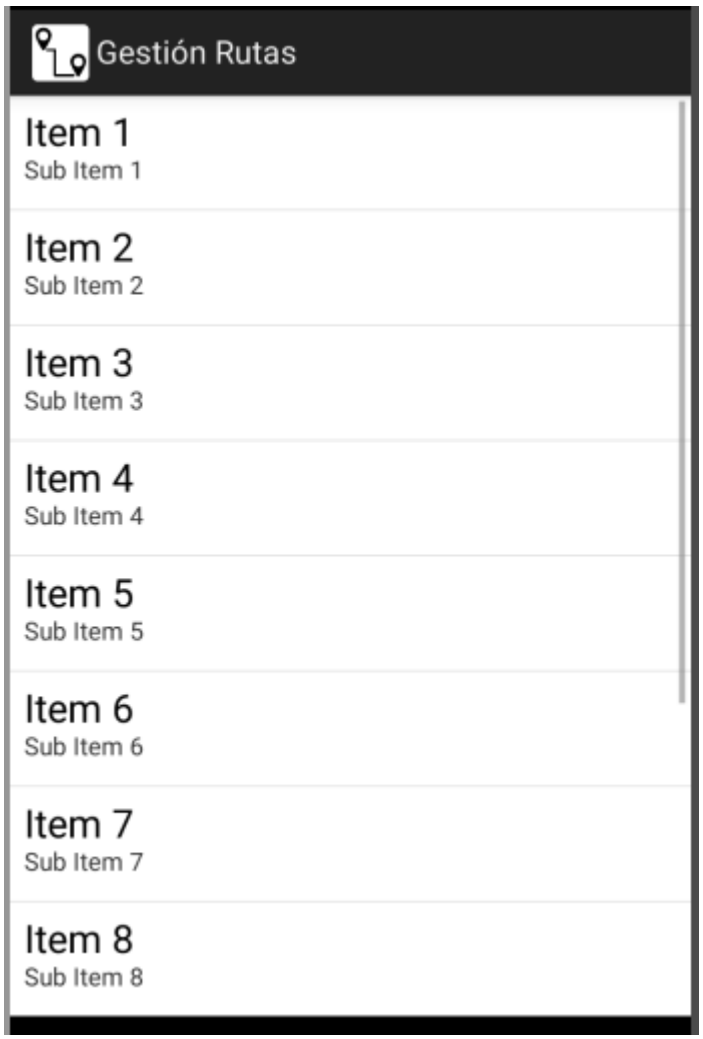


- **HorizontalScrollView:** Tiene la misma funcionalidad que el ViewGroup anterior (ScrollView) pero para la orientación horizontal. Ejemplo:


```
<HorizontalScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/horizontalScrollView" >

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Prueba"
            android:id="@+id/textView6" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Prueba"
            android:id="@+id/textView7" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Prueba"
            android:id="@+id/textView8" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Prueba"
            android:id="@+id/textView5" />

    </LinearLayout>
</HorizontalScrollView>
```

- **ListView**: Sirve para crear una lista con desplazamiento vertical integrado de elementos. Cada uno de los elementos se formará según un formato establecido para ellos (lo habitual es que todos tengan el mismo formato, formando una lista de elementos homogénea) indicado en un fichero de *layout* dado por el sistema o uno personalizado para tal fin. Al utilizar un formato personalizado se debe desarrollar un código Java adicional heredando de la clase *Adapter* correspondiente, en la que haremos uso de ese fichero y asignaremos cada dato a la vista correspondiente de ese *layout*. Ejemplo:

```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView" />
```

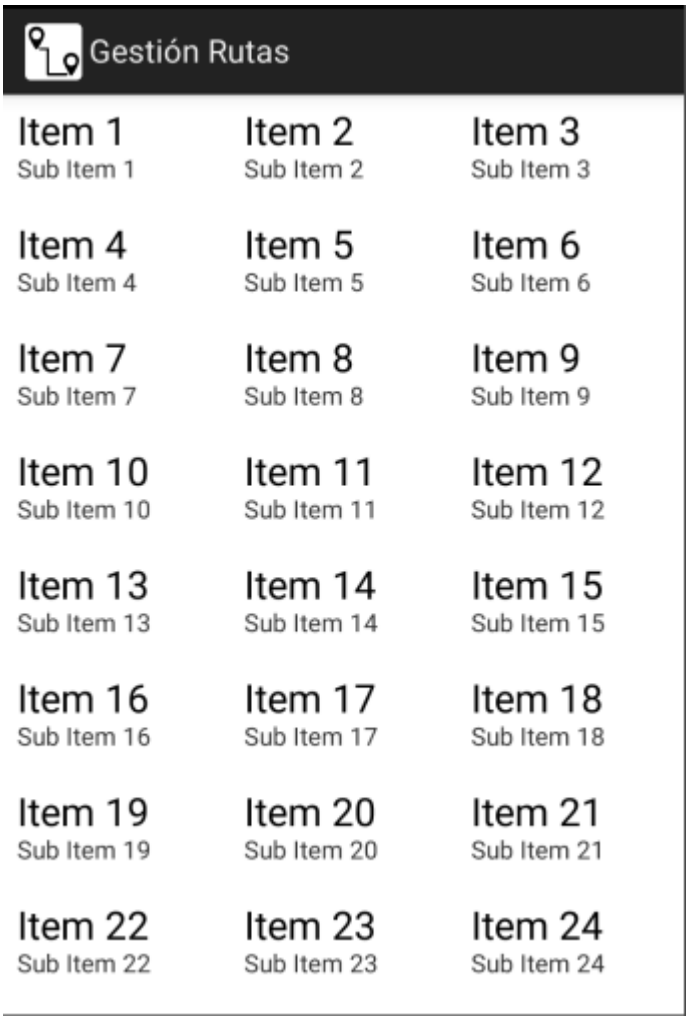


El fichero de *layout* que usará un *Adapter* personalizado puede tener un aspecto similar a éste, y será usado para todos y cada uno de los elementos de ese *ListView*, a no ser que se establezca una lógica adicional en el *Adapter*:

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:visibility="gone">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Prueba"
        android:id="@+id/textView5" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Prueba"
        android:checked="false"
        android:id="@+id/radioButton" />
</LinearLayout>
```

- **GridView**: Sirve para representar en formato de rejilla un conjunto de elementos y dotarlos de un scroll vertical en caso de no poder visualizarse todos en la pantalla del dispositivo. La base del funcionamiento es la misma que la de un *ListView*, salvo que hay que configurar el número de columnas así como la anchura de estas al gusto del desarrollador. De igual manera necesitaremos una clase *Adapter* y un layout nuevo si queremos dotar de alguna personalización extra más allá del formato que nos ofrece el SDK de Android. Ejemplo de un GridView configurado para mostrar tres columnas:

```
<GridView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/gridView"
    android:numColumns="3" />
```



Ejemplo de un fichero de layout para usarlo en un Adapter personalizado:

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView2"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Prueba"
        android:id="@+id/textView9" />
</LinearLayout>
```

3.8. Geolocalización

Android proporciona una API de geolocalización que las aplicaciones pueden utilizar para determinar la ubicación del dispositivo. Para ello necesitaremos importar la librería que Google ha puesto a nuestra disposición y que forma parte del conjunto de librerías Google Play Services, que se llama location:

```
compile 'com.google.android.gms:play-services-location:7.5.0'
compile 'com.google.android.gms:play-services-maps:7.5.0'
compile 'com.google.android.gms:play-services-gcm:7.5.0'
```

En Android se puede utilizar un servicio para que se ejecute y vaya registrando los cambios de ubicación del dispositivo. Este servicio se puede utilizar para llamar al API location y configurar el registro de los cambios de ubicación. Primero es necesario implementar unos métodos del Google Api Client:

```
public class LocalizacionService extends Service implements
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener {
```

```
/**
 * Google api callback methods
 */
@Override
public void onConnectionFailed(ConnectionResult result) {
    Log.i(LOGTAG, "Connection failed: ConnectionResult.getErrorCode() = "
            + result.getErrorCode());
}
```

```
@Override
public void onConnected(Bundle arg0) {
    Log.d(LOGTAG, "----------onConnected");

    // Once connected with google api, get the location
    displayLocation();

    if (mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}
```

```
@Override
public void onConnectionSuspended(int arg0) {
    Log.d(LOGTAG, "----------onConnectionSuspended");
    mGoogleApiClient.connect();
}
```

También es necesario llamar al builder del GoogleApiClient que creará el objeto y lo configurará con unas opciones iniciales, indicándole que queremos hacer uso del LocationServices.API:

```
/**
 * Creating google api client object
 */
protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API).build();
}
```

Lo siguiente que se hará será crear y configurar el LocationRequest que más tarde enviaremos al LocationServices como parámetro:

```
/**
 * Creating google api client object
 */
protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API).build();
}
```

Por último, se necesita definir un listener para definir qué hacer en las distintas situaciones en las que nos responda el servicio de localización. Se define de la siguiente manera, en él se define el método onLocationChanged que será el que se dispare de manera automática al ocurrir un cambio de ubicación:

```
LocationListener mLocationListener = new LocationListener() {

    @Override
    public void onLocationChanged(Location location) {
        // Report to the UI that the location was updated
        String msg = "Updated Location: " +
                Double.toString(location.getLatitude()) + "," +
                Double.toString(location.getLongitude());
        Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();

        // Displaying the new location on UI
        displayLocation();

        //
        if (spiceManager != null) {
            Log.d(LOGTAG, "spicemanager arrancando (onLocationChanged pre): " + spiceManager.isStarted());
            Log.d(LOGTAG, "ruta válida: " + PreferencesManager.getInstance().isRutaValida());

            if (PreferencesManager.getInstance().isRutaValida()) {
                // Assign the new location
                mLastLocation = location;

                executeGuardarPosicion(location, UtilFechas.getFechaActual());
            }
        }
    }
};
```

Una vez configurado el comportamiento que queremos únicamente resta definir dos métodos para arrancar la actualización de cambios en la ubicación y otro para pararla, se hará de la siguiente forma:

```
/**
 * Starting the location updates
 */
protected void startLocationUpdates() {
    LocationServices.FusedLocationApi.requestLocationUpdates(
            mGoogleApiClient, mLocationRequest, mLocationListener);
}
```

```
/**
 * Stopping location updates
 */
protected void stopLocationUpdates() {
    LocationServices.FusedLocationApi.removeLocationUpdates(
            mGoogleApiClient, mLocationListener);
}
```

Al utilizar el Google Api Location en un servicio habrá que asegurarse de arrancar y parar el servicio cuando lo necesitemos desde el código de la aplicación y de definirlo correctamente en el AndroidManifest.xml.

3.9. Comunicación con servicios web

Para llamar a servicios web y recoger su respuesta se puede utilizar las librerías Retrofit y Robospice. Retrofit básicamente se encarga de poner orden el cómo definir el formato de esas peticiones, nos permite definir las llamadas a nuestro API del servidor de manera clara y organizada. Mediante una interface Java y anotaciones, se crean los métodos que representarán a las peticiones de red definidas en el servidor:

```
public interface ApiInterface {

    @FormUrlEncoded
    @POST("/api/loginMovil")
    UsuarioBean login(
            @Field("login") String nombre,
            @Field("clave") String clave
    );
```

Robospice se encarga de gestionar las peticiones en la aplicación y también proporciona un módulo para integrarse con Retrofit. Para realizar esa gestión Robospice utiliza un servicio de Android para realizar las peticiones:

```
public class FollowRetrofitSpiceService extends RetrofitSpiceService {

    //interceptor para los headers
    public static FollowInterceptor followInterceptor = new FollowInterceptor();
    private Converter converter = new GsonConverter(new Gson());

    @Override
    protected Converter createConverter() { return converter; }

    @Override
    public CacheManager createCacheManager(Application application) throws CacheCreationException {
        CacheManager cacheManager = new CacheManager();
        cacheManager.addPersister(new RetrofitObjectPersisterFactory(application, converter, getCacheFolder()));
        return cacheManager;
    }

    @Override
    protected RestAdapter.Builder createRestAdapterBuilder() {
        RestAdapter.Builder restAdapter = super.createRestAdapterBuilder();
        restAdapter.setConverter(converter);
        restAdapter.setLogLevel(RestAdapter.LogLevel.FULL);
        restAdapter.setRequestInterceptor(followInterceptor);
        return restAdapter;
    }

    public File getCacheFolder() { return null; }

    @Override
    protected String getServerUrl() { return FollowRequests.BASE_URL; }
}
```

También facilita el escribir peticiones de red asíncronas de manera óptima y sencilla, se declara el SpiceManager:

```
public class LoginActivity extends Activity {
    private static final String LOGTAG = "LoginActivity";

    protected SpiceManager spiceManager = new SpiceManager(FollowRetrofitSpiceService.class);

    // UI references.
    private AutoCompleteTextView mNombreView;
```

Definimos clase de la petición que vamos a necesitar realizar con los parámetros que necesite pasados en el constructor:

```
public class LoginRequest extends RetrofitSpiceRequest<UsuarioBean, ApiInterface> {

    private String nombre;
    private String clave;

    public LoginRequest() { super(UsuarioBean.class, ApiInterface.class); };

    public LoginRequest(String nombre, String clave) {
        super(UsuarioBean.class, ApiInterface.class);
        this.nombre = nombre;
        this.clave = clave;
    }

    @Override
    public UsuarioBean loadDataFromNetwork() throws Exception {

        return getService().login(this.nombre, this.clave);
    }
}
```

Ejecutamos la petición a través del SpiceManager, haciendo uso del anterior constructor creado para enviarle la información que necesita, y con el “listener” que definamos:

```
// petición de red de login
LoginRequest loginRequest = new LoginRequest(nombre, clave);
spiceManager.execute(loginRequest, new LoginListener());
```

El “listener” recogerá el objeto que devuelve la petición (lo normal será un POJO, aunque también podría ser un simple String), así como cualquier información de error (Http status code y todo el detalle que se reciba):

```
////////////////////////////////////////////////
public final class LoginListener implements RequestListener<UsuarioBean> {
    public final static String LOGTAG = "LoginListener";

    @Override
    public void onRequestFailure(SpiceException spiceException) {
        Log.e(LOGTAG, "error");
        RequestExceptionController.checkSpiceException(LOGTAG, getApplicationContext(), spiceException, spiceManager);
        showProgress(false);
    }

    @Override
    public void onRequestSuccess(UsuarioBean usuarioBean) {

        Log.d(LOGTAG, "usuario: " + usuarioBean.getId() + " " + usuarioBean.getNombre());
        showProgress(false);

        PreferencesManager.getInstance().setUsuario(usuarioBean);
        PreferencesManager.getInstance().setPassword(mClaveView.getText().toString());
        PreferencesManager.getInstance().setLogged(true);
        PreferencesManager.getInstance().clearRuta();

        // vamos al activity principal
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(intent);
    }
}
////////////////////////////////////////////////
```

3.10. Notificaciones PUSH

3.10.1. Introducción

Las notificaciones push son mensaje que se reciben en los dispositivos móviles y que han sido emitidos desde cualquier punto de un sistema. Un ejemplo es una aplicación de cliente de correo, cuando el servidor detecta un mensaje entrante envía una notificación al dispositivo del usuario.

Las notificaciones push permiten el envío de mensaje desde cualquier parte de un sistema a una aplicación móvil, tanto si la aplicación está siendo utilizada por el usuario, está corriendo en un segundo plano, si todavía no ha sido arrancada o, incluso si el dispositivo está en reposo.

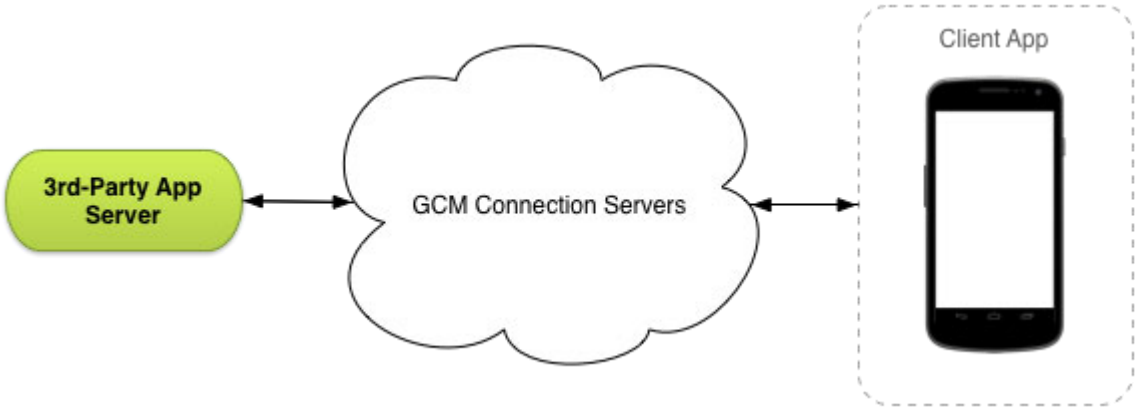
Para ello Google ofrece un servicio de mensajería en la nube: Google Cloud Messaging, que es el que he utilizado en mi proyecto.

Google Cloud Messaging es un servicio gratuito que permite a los desarrolladores enviar mensajes entre servidores y aplicaciones cliente. Esto incluye mensajes desde los servidores a aplicaciones cliente y viceversa, de aplicaciones cliente a servidores. En mi caso solo he desarrollado notificaciones de servidor a cliente.

El servicio GCM maneja todos los aspectos de la cola de mensajes y entrega hacia y desde la aplicación cliente de destino.

3.10.2. Arquitectura

Una implementación de GCM incluye un servidor de Google de conexión, una aplicación de servidor que interactúa con el servidor de conexión a través de HTTP, y una aplicación de cliente.



Estos componentes interactúan del siguiente modo:

- El servidor de Google de conexión, coge los mensajes de la aplicación servidor y los envía a la aplicación cliente.
- En mi aplicación servidor he implementado un protocolo HTTP para comunicarme con el servidor de Google de conexión. La aplicación servidor envía mensajes a servidor de Google de conexión; el servidor de conexión encola y almacena los mensajes y después los envía y después los envía a la aplicación cliente.
- Aplicación cliente recibe las notificaciones.

3.10.3. En la práctica

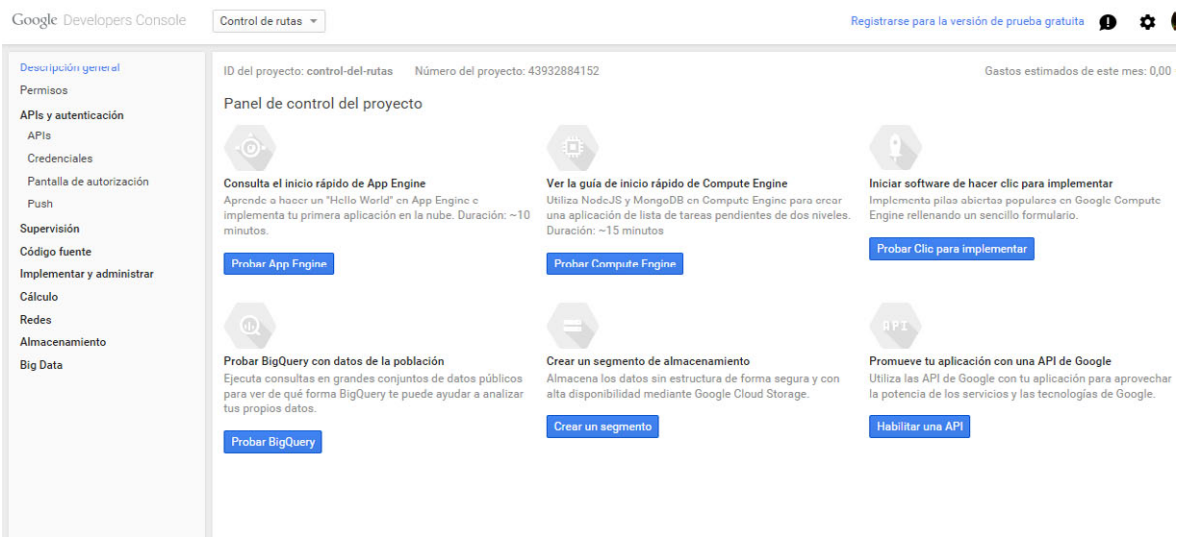
Para poder enviar notificaciones a un dispositivo Android desde GCM es necesario que el dispositivo esté antes registrado.

Lo primero que tengo que hacer para que la aplicación Android pueda recibir notificaciones push desde Google Cloud Messaging es dar de alta mi aplicación en Google Api. Para ello he seguido los siguientes pasos:

Crear un proyecto Google API

Para crear un proyecto en Google API, accedo a la consola de desarrolladores de Google: https://console.developers.google.com y selecciono crear un nuevo proyecto.

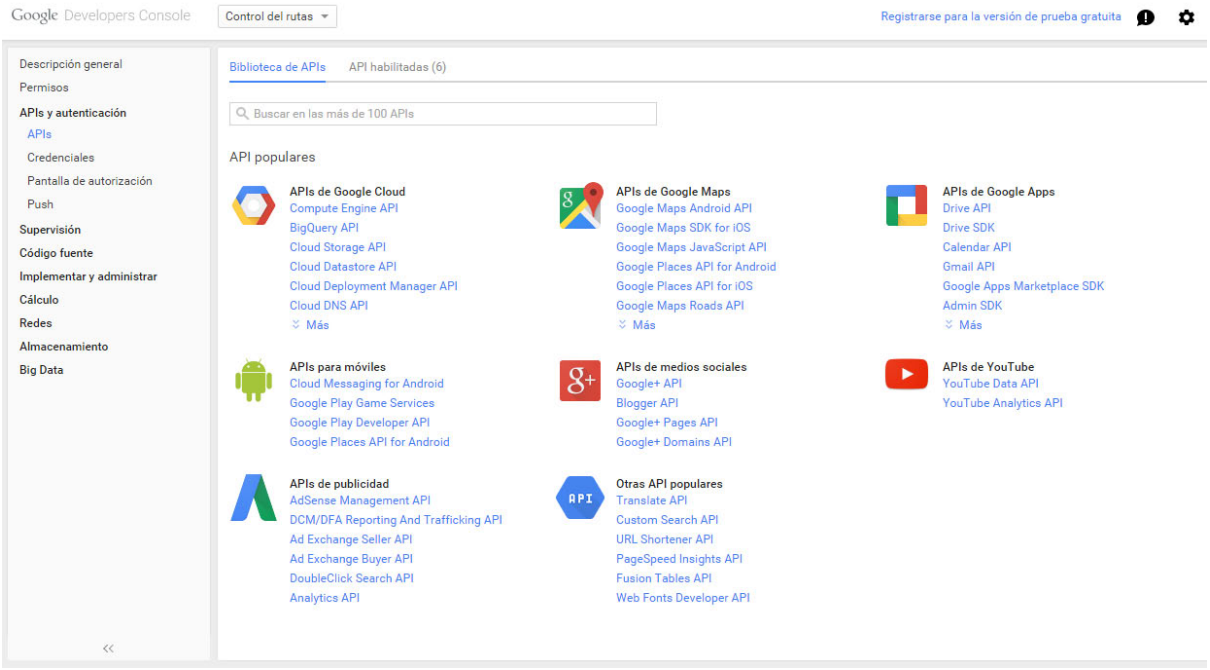
Le asigno un nombre y ya tengo el proyecto creado. Aparece una pantalla donde puedo ver información relevante sobre mi proyecto, como el ID del proyecto y el número del proyecto.

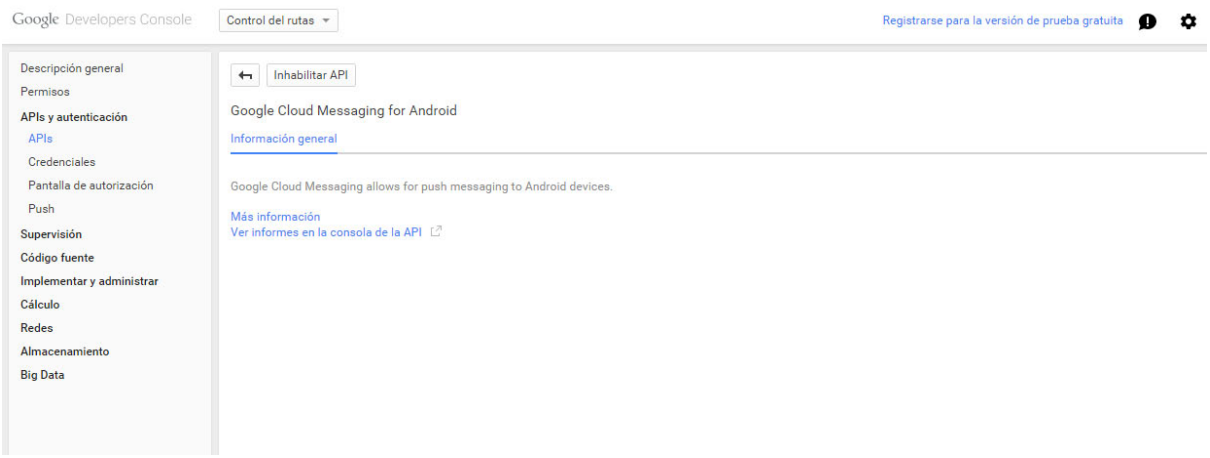


Habilitar Google Cloud Messaging for Android

Para habilitar el servicio Google de conexión, tengo que ir a la opción de API's. En esa opción se encuentran todas las APIs que ofrece la consola de desarrolladores. En mi caso solo me interesa

habilitar la que me permite enviar notificaciones, Cloud Messaging for Android, que se encuentra en la opción de API's para móviles.

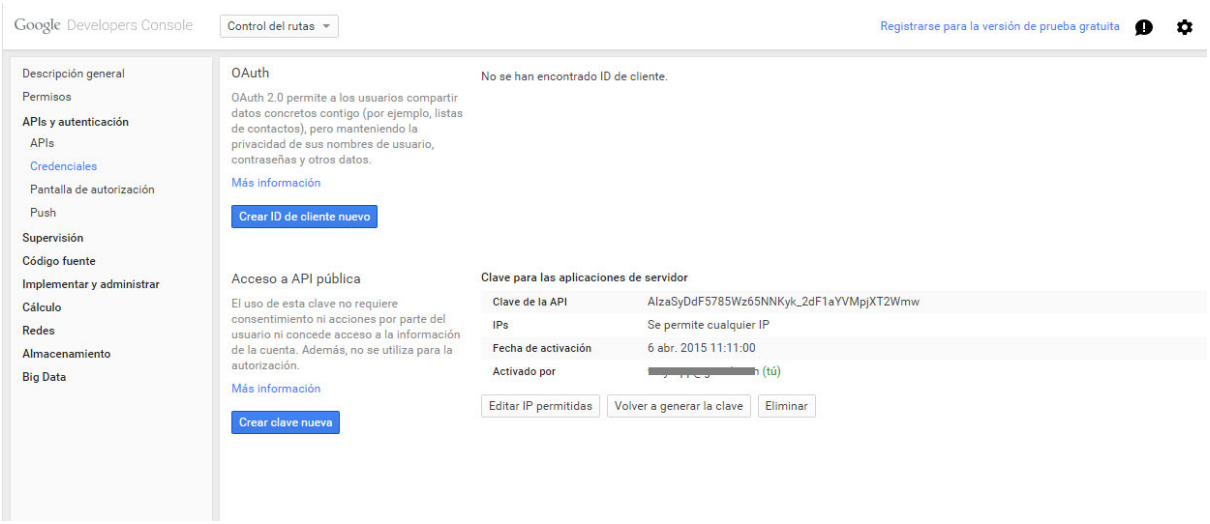




Obtener clave de acceso para la API

Por último, necesito tener una clave de acceso a nuestro API, para poder enviar peticiones a GCM y que éste sirva las notificaciones al dispositivo o dispositivos destino.

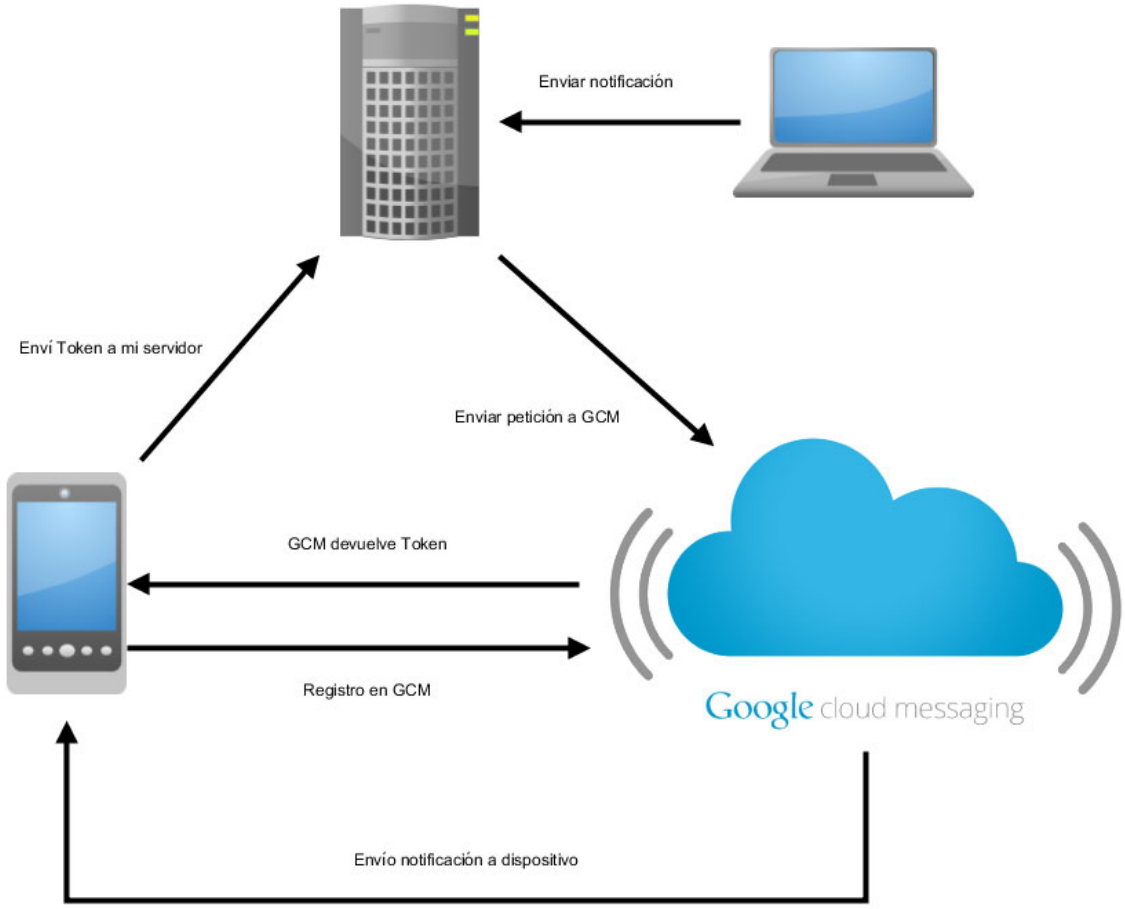
Para generar la clave de acceso, voy a la opción de credenciales y selecciono *Crear clave nueva*, en la opción de Acceso a API pública. Y ya tengo mi clave para conectar a la API.



Registro de dispositivo

Otro paso importante para hacer que mi aplicación Android pueda recibir notificaciones Push desde GCM, es registrar los dispositivos en dicho servicio. Para ello, desde el dispositivo, le indico la aplicación de la que quiero recibir información, indicando el número de proyecto que me asignó la consola de desarrolladores en el paso anterior. De éste modo se le indica a GCM que el dispositivo quiere recibir información de una aplicación concreta.

Flujo completo



3.11. Librerías

Existen varias librerías gratis que nos facilitan el desarrollo en Android. Para mi aplicación he utilizado algunas que detallo a continuación:

- Gson: Esta librería sirve para utilizar de manera sencilla una comunicación mediante JSON con el servidor del sistema. URL: https://github.com/google/gson
- Retrofit: Esta librería sirve para hacer que un API HTTP sea definido de manera sencilla como una interface de Java. URL: http://square.github.io/retrofit/
- Robospice: Esta librería sirve para gestionar las peticiones de red al servidor del sistema. Se definen peticiones y listeners como clases java.

- Robospice-Retrofit: Esta librería sirve de apoyo a la integración de Retrofit con Robospice. Dos de las librerías más usadas para la gestión de las peticiones de red.

4. Diseño del sistema

4.1. Introducción

En este apartado se describirán todos los aspectos relacionados con el diseño de la aplicación, de nombre *Sistema de control de rutas*, detallando cada una de sus funcionalidades y como han sido implementadas.
El principal propósito de este proyecto es desarrollar una aplicación Android que incluya gran parte de las tecnologías descritas anteriormente. El propósito de la aplicación es llevar un seguimiento y control de las rutas de unos vehículos especiales. Para ellos los dispositivos móviles enviarán su posición según las características que se indiquen por configuración, en base a metros y tiempo. También será posible enviar notificaciones a los dispositivos móviles desde la aplicación web.

4.2. Funcionalidad del sistema

4.2.1. Aplicación Web de gestión

La aplicación permite realizar las siguientes funciones:

- Gestión de usuarios. Se podrá gestionar el alta, baja y modificación de los usuarios. Toda la información del usuario será gestionada desde este panel de gestión, salvo el “device token” que será el identificador que se usará para realizar el envío de las notificaciones “push”. Este aspecto lo detallaré más adelante en el apartado “Notificaciones PUSH” de la aplicación Android.

- Consulta de rutas. Se podrá consultar las rutas de los usuarios. Primero se seleccionará un usuario a consultar, se mostrarán todas las rutas de ese usuario ordenadas por fecha y el administrador marcará la ruta a visualizar en el mapa.

- Envío de notificaciones. Se podrá enviar notificaciones a unos o varios dispositivos móviles. Se mostrarán todos los usuarios dados de alta en la aplicación y se podrá elegir uno o todos los usuarios. También aparecerá un campo de texto para introducir el mensaje de la notificación.

4.2.2. Aplicación Android

La aplicación permite realizar las siguientes funciones:

- Login
- Comienzo ruta. El servidor creará un id nuevo de ruta (varias comprobaciones) y se lo devolverá al dispositivo móvil.
- Fin ruta. Se almacenará en el servidor la última posición recibida y se cerrará la ruta.
- Recepción de mensajes por notificación push desde la aplicación web.
- En segundo plano la aplicación realizará:
    - Envío de posicionamiento al servidor
    - Petición a Google del device token
    - Envío al servidor del device token proporcionado por Google.
    - Recepción de las notificaciones push y la interpretación de éstas.

4.2.3. Notificaciones Push

En este apartado voy a detallar como he desarrollado el envío de notificaciones desde la aplicación web al dispositivo móvil, mediante un método Http Post, y como la aplicación Android las recibe y procesa.

PARTE WEB

Lo primero es declarar dos constantes, una con la clave de acceso para la API y otra con la url de la API de envío de Google Cloud Message. La obtención de la clave de acceso la detallé en el punto 3.9.3, apartado **Obtener clave de acceso para la API**:

```
private static String API_KEY = "AIzaSyDdF5785Wz65NNKyk_2dF1aYVMpjXT2Wmw";
private static String URL_GCM = "https://android.googleapis.com/gcm/send";
```

Por otro lado, necesito construir la notificación en un JSON, con los campos mínimos que requiere la API de Google, los device token que recibirán la notificación, que explicaré más abajo cómo se obtienen, y el mensaje que recibirá la aplicación móvil y que tiene que ir encapsulada en una nodo “data”:

```
{
    "registration_ids": [
        "111",
        "222",
        "333",
        "444"
    ],
    "data": {
        "message": "este es el mensaje"
    }
}
```

```
private static String construirJson(Usuario usu, String msg){

    ObjectNode mainData = Json.newObject();
    ObjectNode data = Json.newObject();
    ArrayList<String> arr = new ArrayList<String>();

    //Añadir token de usuario
    arr.add(usu.token);

    //Ids destinatarios
    JsonNode node = Json.toJson(arr);
    mainData.put("registration_ids", node);

    //Mensaje
    data.put("message", msg);
    mainData.put("data", data);

    return mainData.toString();
}
```

Por último el método que realiza el envío, del Json anterior, y trata la respuestas de GCM. Las respuesta de GCM puede ser de dos tipos:

- El mensaje se ha procesado correctamente. La respuesta HTTP tiene un estado 200, y el cuerpo contiene más información acerca del estado del mensaje.
- El servidor de GCM rechaza la solicitud. La respuesta HTTP contiene un código de estado no-200 (por ejemplo, 400, 401 o 5xx).

```
public static Result enviarNotificacion(){

    ObjectNode result = Json.newObject();
    Usuario usu = null;
    Long idUsuario;
    String msg = "";
    boolean mensajeEnviado = false;

    HttpClient client = new DefaultHttpClient();
    HttpPost post = new HttpPost(URL_GCM);
    ObjectNode mainData = Json.newObject();
    ObjectNode data = Json.newObject();
    HttpResponse response = null;

    DynamicForm formulario = Form.form().bindFromRequest();

    //Recuperar datos formulario
    idUsuario = Long.parseLong(formulario.get("id_usuario"));
    msg = formulario.get("msg");

    usu = Usuario.find.where().idEq(idUsuario).findUnique();

    StringEntity se = null;

    String mensaje = construirJson(usu, msg);

    try {
        se = new StringEntity(mensaje);
    } catch (UnsupportedEncodingException e2) {
        Logger.error("Error " + e2.toString());
    }
    post.setEntity(se);
    post.addHeader("Authorization", "key=" + API_KEY);
    post.addHeader("Content-Type", "application/json");


    //Envio mensaje a GCM
    Logger.info("Comienzo envio mensaje. ");
    try {
        response = client.execute(post);
    } catch (ClientProtocolException e2) {
        Logger.error("Error de protocolo al enviar el mensaje: " + e2.toString());
    } catch (IOException e2) {
        Logger.error("Error al enviar el mensaje: " + e2.toString());
    }
    Logger.info("Fin envio mensaje. ");
    //Fin envio mensaje
```

```
        //Tratar respueta de GCM
        BufferedReader rd = null;
        StringBuffer respuesta = new StringBuffer();
        int code = 0;
        String line = "";

        try {
            Logger.info("response code = " + Integer.toString(response.getStatusLine().getStatusCode()));
            code = response.getStatusLine().getStatusCode();
            rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

            while ((line = rd.readLine()) != null)
            {
                respuesta.append(line);
            }

        } catch (IllegalStateException e1) {
            Logger.error("Error IllegalStateException " + e1.toString());
        } catch (IOException e1) {
            Logger.error("Error al tratar el mensaje de respuesta " + e1.toString());
        }

        Logger.info("La respuesta ha sido: " + respuesta.toString());

        if(HttpStatus.SC_OK == code){
            flash("success", "El mensaje se ha enviado con existo");
        }else{
            flash("error", "El mensaje no se ha enviado con existo");
            Logger.error("El código de respuesta ha sido: " + code);
        }

        return redirect("/notificaciones/elegirUsuario");

    }
```

PARTE ANDROID

Lo primero para que el servidor de aplicaciones y la aplicación cliente puedan intercambiar mensajes es completar un “apretón de manos”. En este proceso, el cliente obtiene una ficha de registro único y lo pasa a la aplicación web, que almacena el token y envía un acuse de recibo de vuelta a la aplicación cliente. El token de registro obtenido en este proceso es el identificador que utiliza la aplicación web para enviar mensajes al cliente.

Primero la aplicación cliente solicita token de registro a Google utilizando el API ID de instancia:

```
 */
public static void registerInBackground() {

    new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            String msg = "";
            try {
                if (mGcm == null) {
                    mGcm = GoogleCloudMessaging.getInstance(mActivity.getApplicationContext());
                }
                mRegId = mGcm.register(SENDER_ID);
                msg = "Device registered, registration ID=" + mRegId;

                // Persist the regID - no need to register again.
                PreferencesManager.getInstance().setGcmRegId(mRegId);

            } catch (IOException ex) {
                msg = "Error :" + ex.getMessage();
                // If there is an error, don't just keep trying to register.
                // Require the user to click a button again, or perform
                // exponential back-off.
            }
            return msg;
        }

        @Override
        protected void onPostExecute(String msg) {
            Log.d(LOGTAG, msg);
            //Toast.makeText(mActivity.getApplicationContext(), "onPostExecute----> " + msg, Toast.LENGTH_SHORT).show();
            // You should send the registration ID to your server over HTTP, so it
            // can use GCM/HTTP or CCS to send messages to your app.
            enviarRegIdaServidor(mRegId);
        }

        @Override
        protected void onCancelled() { super.onCancelled(); }
    }.execute(null, null, null);
}
```

Con anterioridad, he habilitado dicha API, Cloud Messaging for Android en la consola de desarrolladores y he obtenido un API ID. Estos pasos están detallados en el apartado 3.9.3, punto **Habilitar Google Cloud Messaging for Android** y **Obtener clave de acceso para la API**.

La aplicación Android pasa el token de registro a la aplicación web:

```
/**
 * Sends the registration ID to your server over HTTP, so it can use GCM/HTTP or CCS to send
 * messages to your app. Not needed for this demo since the device sends upstream messages
 * to a server that echoes back the message using the 'from' address in the message.
 */
private static void enviarRegIdaServidor(String token) {
    // Your implementation here.
    Log.d(LOGTAG, "GCMUtils >>>>>>>>>>>>>>>>>>>>>>>>> deviceToken: " + token);

    if (!token.isEmpty()) {
        // petición de red de para enviar el device token al servidor
        RegistrarDispositivoRequest request = new RegistrarDispositivoRequest(PreferencesManager.getInstance().getUsuario().getId(), token);
        ((MainActivity) mActivity).getSpiceManager().execute(request, null);
    }
}
```

Por último la aplicación web guarda el token de registro e informa a la aplicación Android que proceso finalizó correctamente.

```
public static Result registrarDispositivo(){

    Usuario usu = new Usuario();

    String idUsuario;
    ObjectNode result = Json.newObject();

    DynamicForm formulario = Form.form().bindFromRequest();

    if(formulario.get("id_usuario") != null && !formulario.get("id_usuario").isEmpty()){
        idUsuario = formulario.get("id_usuario");

        usu = Usuario.find.where().idEq(idUsuario).findUnique();

        if(usu == null){
            result.put("status", "El usuario no existe");
            return notFound(result);
        }else{
            if(formulario.get("token") != null){
                usu.token = formulario.get("token");
            }
        }
    }else{
        result.put("status", "Falta usuario");
        return badRequest(result);
    }

    usu.save();

    result.put("status", "OK");

    return ok(result);

}
```

Si no se pudiera terminar el registro del token correctamente. La aplicación Android volverá a intentarlo varias veces y si no descartará el token.

Como paso previo a poder utilizar las notificaciones push de GCM, hay que configurar la aplicación para tal fin, añadiendo en el fichero AndroidManifest.xml las líneas:

```
<!-- GCM BEGIN -->
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

<permission
    android:name="vero.es.follow.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />

<uses-permission android:name="vero.es.follow.permission.C2D_MESSAGE" />
<!-- GCM END -->
```

```
<!-- GCM BEGIN -->
<receiver
    android:name="vero.es.follow.gcm.GcmBroadcastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />

        <category android:name="vero.es.follow" />
    </intent-filter>
</receiver>

<service android:name="vero.es.follow.gcm.GcmIntentService" />

<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
<!-- GCM END -->
```

Y a su vez declarar las dos clases que se encargarán de permanecer a la escucha para recoger las notificaciones push y construir la notificación con la información recibida, más concretamente en los siguientes extractos de código:

```
// Put the message into a notification and post it.
// This is just one simple example of what you might choose to do with
// a GCM message
private void sendNotification(String title, String msg, int type) {
    Log.d(LOGTAG, "sendNotification");

    NotificationManager notificationManager = (NotificationManager)
            this.getSystemService(Context.NOTIFICATION_SERVICE);
    NotificationCompat.Builder mBuilder = buildNotification(title, msg, type);
    notificationManager.notify(type + "", type, mBuilder.build());
}
```

```
private NotificationCompat.Builder buildNotification(String title, String msg, int type) {
    Log.d(LOGTAG, "buildNotification");

    Intent notificationIntent = new Intent(getApplicationContext(), MainActivity.class);
    notificationIntent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP | Intent.FLAG_ACTIVITY_CLEAR_TOP);
    PendingIntent pendingIntent = PendingIntent.getActivity(
            getApplicationContext(),
            0,
            notificationIntent,
            PendingIntent.FLAG_UPDATE_CURRENT);

    int notDefaults = Notification.DEFAULT_SOUND;
    notDefaults |= Notification.DEFAULT_VIBRATE;

    NotificationCompat.Builder builder =
            new NotificationCompat.Builder(this)
                    .setSmallIcon(R.drawable.ic_launcher)
                    .setContentTitle(title)
                    .setStyle(new NotificationCompat.BigTextStyle()
                            .bigText(msg))
                    .setDefaults(notDefaults)
                    .setContentText(msg);

    builder.setContentIntent(pendingIntent);
    builder.setAutoCancel(true);

    return builder;
}
```

```
public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {

    private static final String LOGTAG = "GcmBroadcastReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {

        boolean logged = PreferencesManager.getInstance().getLogged();

        if (logged) {
            Log.d(LOGTAG, "logueado");
            // Explicitly specify that GcmIntentService will handle the intent.
            ComponentName comp = new ComponentName(context.getPackageName(),
                    GcmIntentService.class.getName());
            // Start the service, keeping the device awake while it is launching.
            startWakefulService(context, (intent.setComponent(comp)));
            setResultCode(Activity.RESULT_OK);
        } else {
            Log.d(LOGTAG, "no logueado -> no hacer nada");
        }
    }
}
```

4.2.4. API Google Maps

En este apartado voy a hacer un breve introducción a la API de Google Maps y como la he utilizado para dibujar las rutas de los dispositivos móviles.

Lo primero es incluir la API de Google Maps JavaScript.Esta librería se pueden descargar al proyecto o acceder a ella vía una URL al cargar la página. Para ello en mi página tendré que añadir la llamada a esa URL:

<script src="**https://maps.googleapis.com/maps/api/js**"></script>

Esta solicitud de arranque carga todos los principales objetos de Javascript y símbolos necesarios para usar en la API de Google Maps. No todas las funciones de la API de Google Maps se cargan directamente con esa llamada, hay otros componentes complementarios que solo se cargan bajo demanda. Esto permite a la API principal cargar de forma más rápida. Una vez cargadas se accede vía *google.map.NombreLibreria*.

Para soportar el mapa es necesario crear en la página HTML un elemento contenedor, como es un capa:

A continuación se crea una función JavaScript para inicializar el mapa y calcular la ruta. Esta función se llamará al cargar la ventana. Es esta función se crea el objeto map y se inicializa con un zoom y una posición central y se asigna al elemento div que he creado en la página HTML. Por último se inicializa un escuchador de eventos, donde se le pasa el mapa, el estado y una función para en mi casa obtener la ruta de las posiciones pasada a la página.

```
function initialize() {

    //Función para completar los campos de fechas de la página
    fillInDates();

    var inicio = new google.maps.LatLng(40.347539, -3.809847);
    var mapOptions = {
        zoom : 10,
        center : inicio
    }

    // Crea mapa con las opciones anteriores sobre el elemento div
    map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);

    // Inicializa el escuchador de eventos, en modo ocioso
    // y llamo función principal para crear ruta
    google.maps.event.addListenerOnce(map, 'idle', function() {
        main();
    });

}
```

Para dibujar la ruta, primero hay que calcular las direcciones de las posiciones que tengo almacenadas. Para calcular las direcciones hay que utilizar el objeto DirectionsService, que se comunica con la API Directions Service de Google Maps. Las direcciones pueden especificar orígenes y destinos, ya sea como cadenas de texto o como valores LatLng. El servicio puede devolver direcciones de varias partes utilizando una serie de waypoints (puntos intermedios). Las direcciones se muestran como una polilínea dibujando la ruta en un mapa, o adicionalmente como una serie de descripción textual, dentro de un elemento .

El acceso al servicio DirectionsService es asíncrono, ya que el API de Google Maps tiene que hacer una llamada a un servidor externo. Por esta razón, es necesario pasar un método de retorno de llamada que se ejecutará una vez finalizada la petición. Este método procesará el resultado.

En este punto, me encuentro con un problema: no puedo dibujar una ruta con más de 8 puntos intermedios de manera automática con la función que proporciona Google Maps. La licencia gratuita no lo permite y la de pago tampoco soluciona mi problema porque solo permite 23 y mi aplicación puede almacenar ruta mayores, aunque esto siempre va a depender de la configuración de la captura de posiciones, pero si quiero obtener las rutas lo más exactas posibles, cuantas más posiciones almacene mayor es la exactitud del trayecto del dispositivo móvil.

Para solucionar este problema he utilizado una función recursiva, donde se concatenan varias direcciones de 10 puntos entre si. A esta función recursiva se le pasan todas las posiciones de una ruta. Por cada vuelta se recuperan 10 posiciones, inicio, fin y 8 puntos intermedios y se dibuja una línea que se concatena a la línea obtenida en cada vuelta, de este modo se consigue una ruta de más de 10 posiciones.

```
function gDirRequest(directionsService, waypoints, userFunction, waypointIndex, path) {

    // Comprobar si es la prinera vuelta y están inicializados la ruta y el índice
    // para recorrer el array de posiciones
    waypointIndex = typeof waypointIndex !== 'undefined' ? waypointIndex : 0;
    path = typeof path !== 'undefined' ? path : [];

    // Obtener las 10 siguientes posicionees
    var s = gDirGetNextSet(waypoints, waypointIndex);

    // construir objeto DirectionsRequest
    var startl = s[0].shift()["location"];
    var endl = s[0].pop()["location"];
    var request = {
        origin : startl,
        destination : endl,
        waypoints : s[0],
        travelMode : google.maps.TravelMode.DRIVING, //Ruta por carretera
        unitSystem : google.maps.UnitSystem.METRIC,
        optimizeWaypoints : false,
        provideRouteAlternatives : false,
        avoidHighways : false,
        avoidTolls : false
    };

    // Acceso al servicio de Direcciones
    directionsService.route(request, function(response, status) {

        if (status == google.maps.DirectionsStatus.OK) {
            path = path.concat(response.routes[0].overview_path);
            oldpath = path
            if (s[1] != null) {
                lastIndx = s[1]
                gDirRequest(directionsService, waypoints, userFunction, s[1], path)
            } else {
                userFunction(path);
            }

        } else {
            path = oldpath;
            lastIndx = lastIndx + 1
            if (s[lastIndx] != null) {
                gDirRequest(directionsService, waypoints, userFunction, lastIndx, path)
            } else {
                userFunction(path);
            }
        }

    }// Fin función tratar respuesta de la llamada al servicio
    );
}
```

```
function gDirGetNextSet(waypoints, startIndex) {
    var MAX_WAYPOINTS_PER_REQUEST = 8;

    var w = []; // array de waypoints a devolver

    if (startIndex > waypoints.length - 1) {
        return [ w, null ];
    } // no mas waypoints para procesar

    var endIndex = startIndex + MAX_WAYPOINTS_PER_REQUEST;

    endIndex += 2;

    // controlar que no me paso
    if (endIndex > waypoints.length - 1) {
        endIndex = waypoints.length;
    }

    for (var i = startIndex; i < endIndex; i++) {
        w.push(waypoints[i]);
    }

    if (endIndex != waypoints.length) {
        return [ w, endIndex -= 1 ];
    } else {
        return [ w, null ];
    }
}
```

La API de Google Maps también permite crear y configurar marcadores, que sirven para identificar una localización en el mapa. Los marcadores se construyen con el constructor google.maps.Marker, en el cual se pueden inicializar una serie de propiedades.

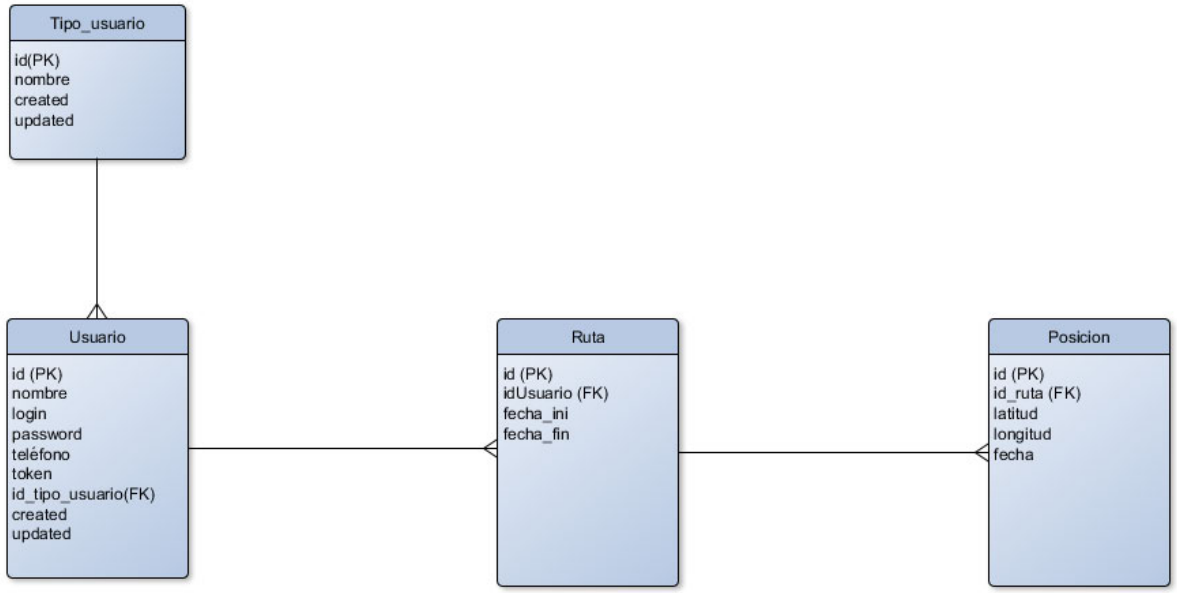
En mi caso he necesitado crear dos marcadores, uno para indicar el comienzo de la ruta y otro para indicar el final de la ruta, cuando corresponda. Para ello he creado una función donde les paso una posición y un título, según el marcado que quiera construir.

```
function addMarker(positionMarker, titleMarker) {

    var marker = new google.maps.Marker({
        position : positionMarker,
        map : map,
        title : titleMarker,
    });

    marker.setMap(map);
}
```

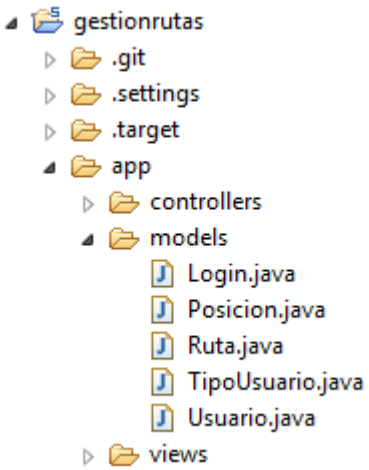
4.3. Modelo de datos

El diseño de la aplicación web parte inicialmente de definir el modelo de datos y en función de ese modelo de datos he desarrollado el patrón MVC.

Diagrama Entidad-Relación



Después de tener claro mi modelo de datos, paso a implementarlo en Eclipse. Creo una clase por cada entidad, con todos sus atributos.



NOTA: La clase Login no forma parte del modelo...

Defino las clases como entidades, @Entity, y directamente el framework me creará las tablas en BBDD.

Clase TipoUsuario

```
@Entity
public class TipoUsuario  extends Model{

    @Id
    public Long id;

    @Required(message= "Debe introducir un nombre")
    public String nombre;

    @DateTime(pattern="dd-MM-yyyy - hh:mm")
    public Date created;

    @DateTime(pattern="dd-MM-yyyy - hh:mm")
    public Date updated;
```

Clase Usuario

```
@Entity //persitido en BBDD
public class Usuario  extends Model{

    @Id
    public Long id;

    @Required(message= "Debe introducir un nombre")
    public String nombre;

    @Required(message= "Debe introducir el login")
    public String login;

    @Required(message= "Debe introducir el password")
    public String password;

    @Required(message= "Debe introducir un número de teléfono")
    public String telefono;

    public String token;

    @ManyToOne
    @Required(message= "Debe seleccionar un tipo de usuario")
    public Long idTipoUsuario;

    @DateTime(pattern="dd-MM-yyyy - hh:mm")
    public Date created;

    @DateTime(pattern="dd-MM-yyyy - hh:mm")
    public Date updated;
```

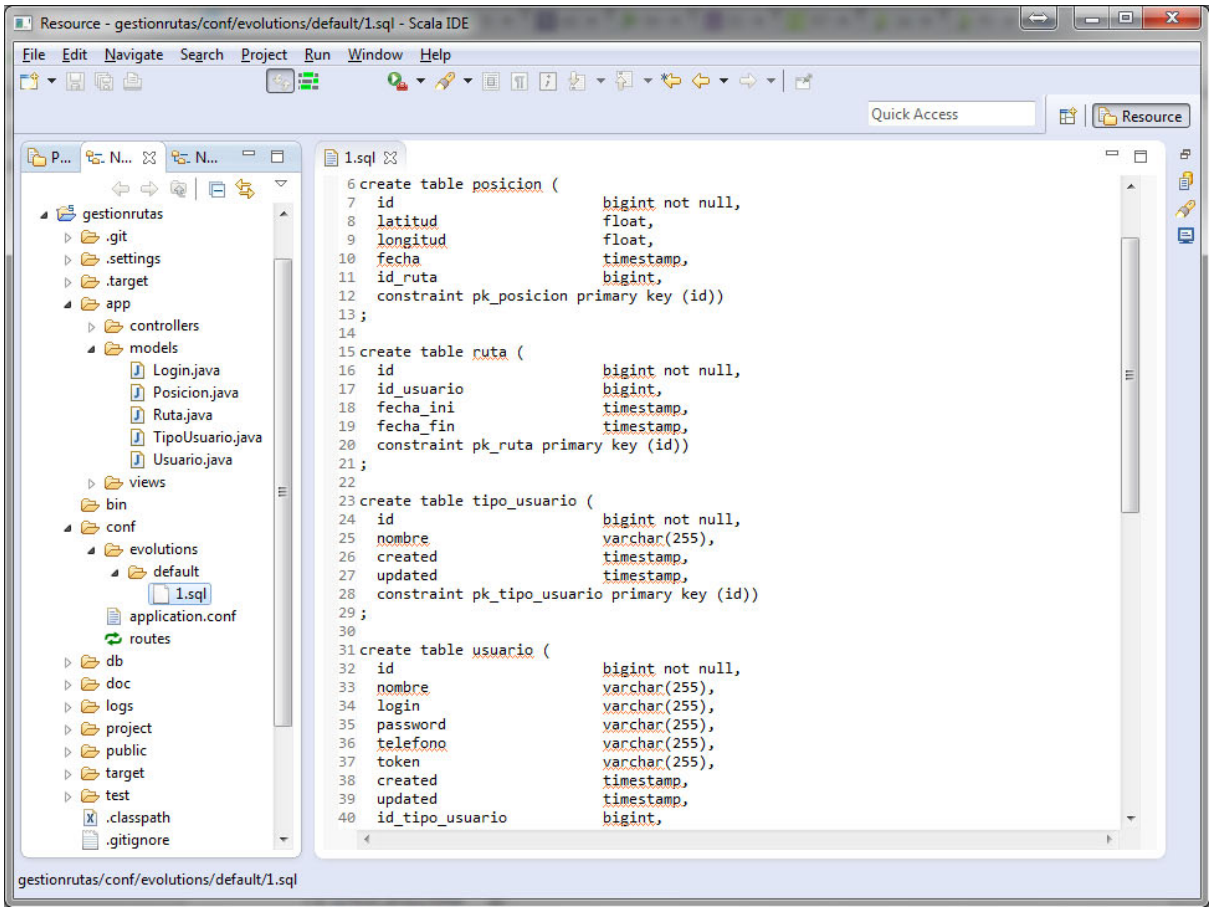
Clase Ruta

```
@Entity //persitido en BBDD
public class Ruta  extends Model{

    @Id
    public Long id;

    @ManyToOne
    @Required(message= "Requerido un usuario")
    public Long idUsuario;

    @DateTime(pattern="yyyy-MM-dd")
    @Required(message= "Requerida una fecha")
    public Date fechaIni;

    @DateTime(pattern="yyyy-MM-dd")
    @Required(message= "Requerida una fecha")
    public Date fechaFin;
```

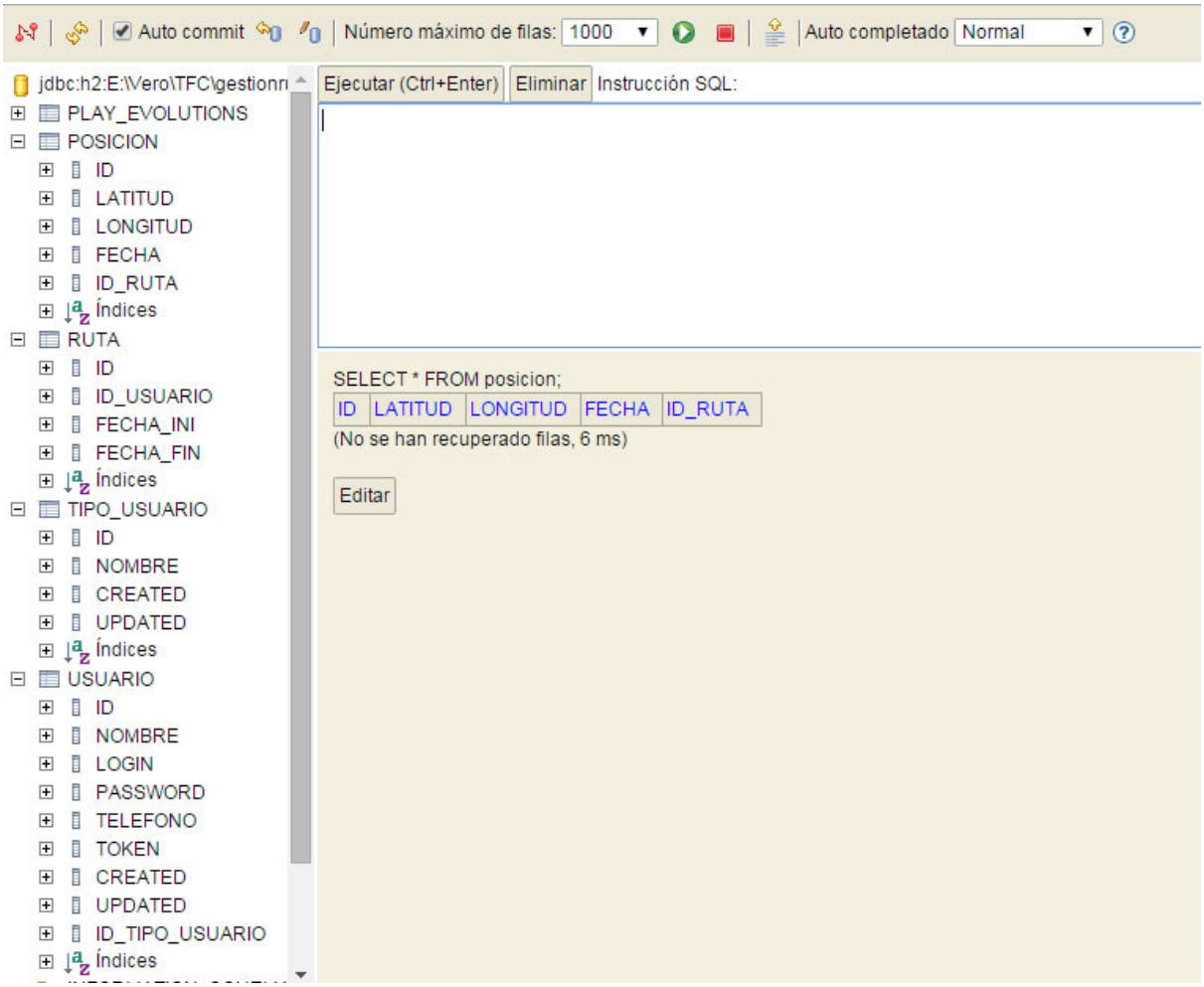
Clase Posición

```
@Entity //persitido en BBDD
public class Posicion  extends Model{

    @Id
    public Long id;

    @Required(message= "Requerida la latitud")
    public Float latitud;

    @Required(message= "Requerida la longitud")
    public Float longitud;

    @DateTime(pattern="dd-MM-yyyy - hh:mm:ss")
    @Required(message= "Requerida una fecha")
    public Date fecha;

    @ManyToOne
    @Required(message= "Requerida una ruta")
    public Long idRuta;
```

Cuando compilo se genera un fichero con las instrucciones sql para crear las tablas en base de datos. Este fichero se crea en el directorio de configuración, con el nombre 1.sql. Si hiciera modificaciones en mi modelo de datos, se crearían más ficheros de ese tipo.

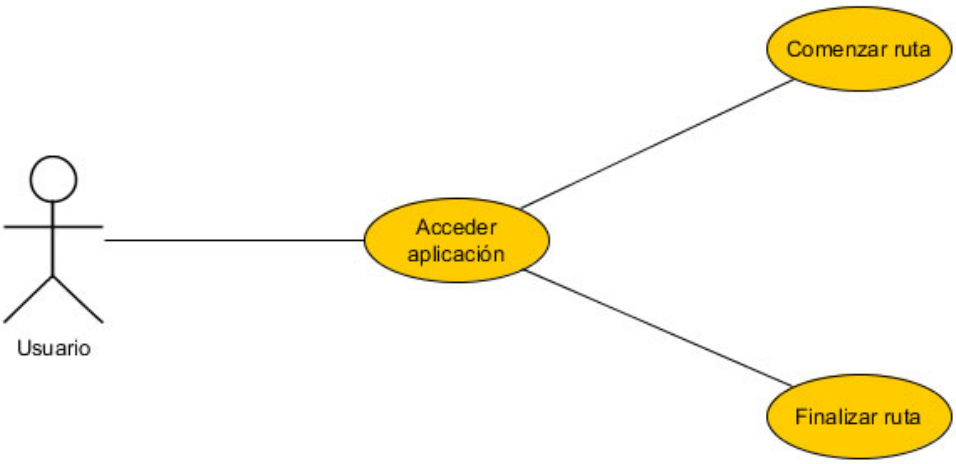

Resource - gestionrutas/conf/evolutions/default/1.sql - Scala IDE
File Edit Navigate Search Project Run Window Help
Quick Access
Resource
gestionrutas
.git
.settings
.target
app
controllers
models
Login.java
Posicion.java
Ruta.java
TipoUsuario.java
Usuario.java
views
bin
conf
evolutions
default
1.sql
application.conf
routes
db
doc
logs
project
public
target
test
.classpath
.gitignore
1.sql
create table posicion (
id bigint not null,
latitud float,
longitud float,
fecha timestamp,
id_ruta bigint,
constraint pk_posicion primary key (id))
;
create table ruta (
id bigint not null,
id_usuario bigint,
fecha_ini timestamp,
fecha_fin timestamp,
constraint pk_ruta primary key (id))
;
create table tipo_usuario (
id bigint not null,
nombre varchar(255),
created timestamp,
updated timestamp,
constraint pk_tipo_usuario primary key (id))
;
create table usuario (
id bigint not null,
nombre varchar(255),
login varchar(255),
password varchar(255),
telefono varchar(255),
token varchar(255),
created timestamp,
updated timestamp,
id_tipo_usuario bigint,
gestionrutas/conf/evolutions/default/1.sql

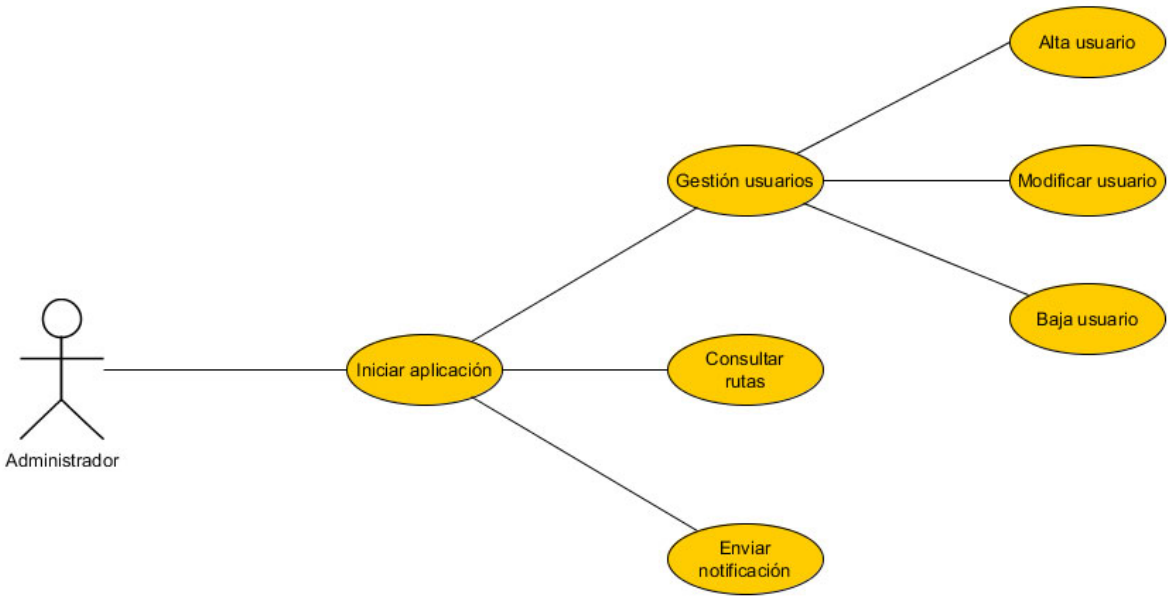
En la base de datos se crean, también automáticamente, las tablas según el script generado al definir las clases entidad.



4.4. Casos de uso

Una vez presentada la aplicación se procederá a exponer los casos de uso. A continuación muestro los actores del proyecto y las acciones permitidas según la aplicación con la que interactúan, aplicación Android o aplicación Web:





4.4.1. Detalle de los casos de uso

Nombre	Acceder aplicación
Descripción	Abrir la aplicación desde el menú de aplicaciones.
Actores	Usuario
Precondiciones	Haber instalado la aplicación correctamente. Tener internet. Estar dado de alta en la aplicación web.
Excepciones	Si no está dado de alta no podrá entrar en la aplicación
Postcondiciones	Ninguna

Nombre	Comenzar ruta
Descripción	Indicar que se comienza una ruta.
Actores	Usuario
Precondiciones	No tener otra ruta iniciada. Tener conexión a internet.
Excepciones	Ninguna
Postcondiciones	Ninguna

Nombre	Finalizar ruta
Descripción	Indicar que finaliza una ruta.
Actores	Usuario
Precondiciones	Haber iniciado una ruta.
Excepciones	Ninguna
Postcondiciones	Ninguna

Nombre	Iniciar aplicación
Descripción	Acceder con usuario y contraseña a la aplicación web
Actores	Administrador
Precondiciones	Ninguna
Excepciones	Ninguna
Postcondiciones	Ninguna

Nombre	Gestionar usuarios
Descripción	Muestra una lista de usuarios y las opciones para su gestión
Actores	Administrador
Precondiciones	Ninguna
Excepciones	Ninguna
Postcondiciones	Ninguna

Nombre	Alta usuario
Descripción	Muestra un formulario para dar de alta un usuario nuevo
Actores	Administrador
Precondiciones	Tener permiso
Excepciones	Falta completar algún dato obligatorio o el usuario ya existe en base de datos
Postcondiciones	Ninguna

Nombre	Modificar usuario
Descripción	Muestra un formulario con los datos del usuario para modificar lo que se desee.
Actores	Administrador
Precondiciones	Ninguna
Excepciones	Falta completar algún dato obligatorio
Postcondiciones	Ninguna

Nombre	Baja usuario
Descripción	Elimina de base de datos el usuario seleccionado.
Actores	Administrador
Precondiciones	Ninguna
Excepciones	Ninguna
Postcondiciones	Ninguna

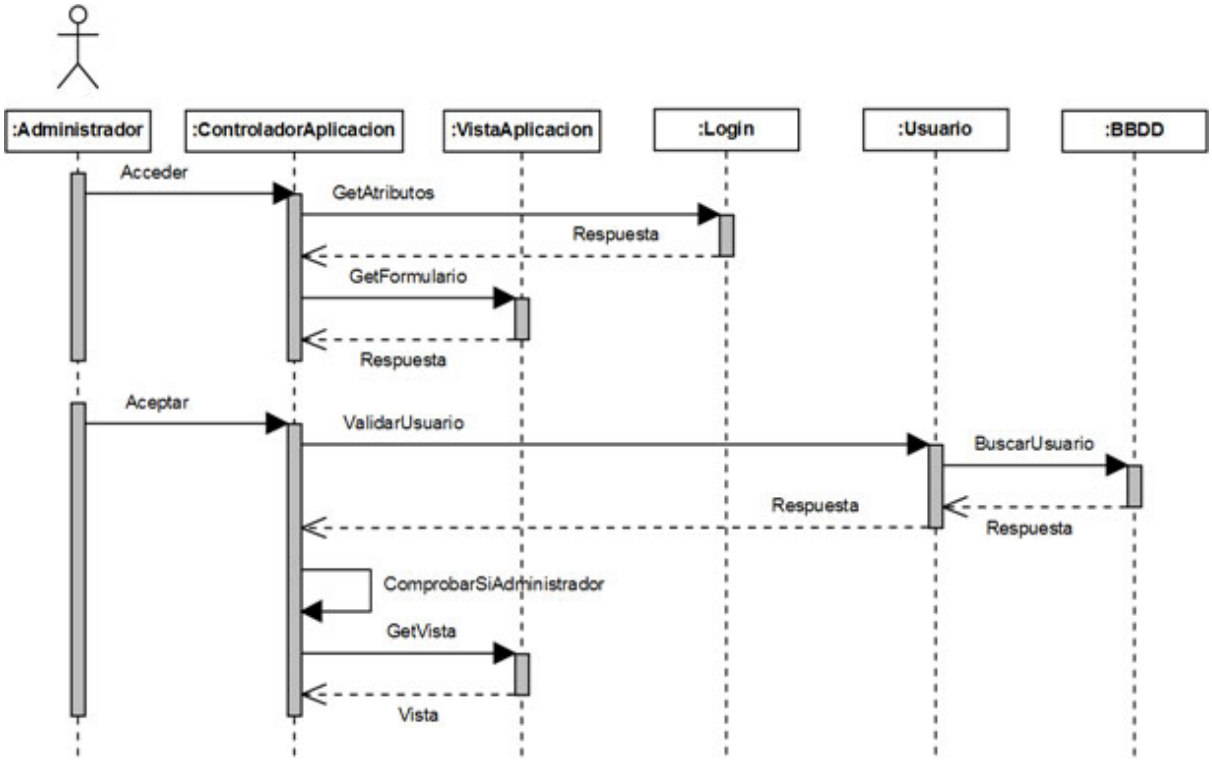
Nombre	Consultar rutas
Descripción	Se muestra una lista de usuario para seleccionar y luego se muestra una lista de sus rutas ordenada por fecha y hora.
Actores	Administrador
Precondiciones	Usuario con rutas
Excepciones	Ninguna
Postcondiciones	Ninguna

Nombre	Enviar notificación
Descripción	Envía una notificación al dispositivo del usuario seleccionado
Actores	Administrador
Precondiciones	Dar de alta aplicación en Google API, registrar dispositivos móviles en GCM
Excepciones	
Postcondiciones	Ninguna

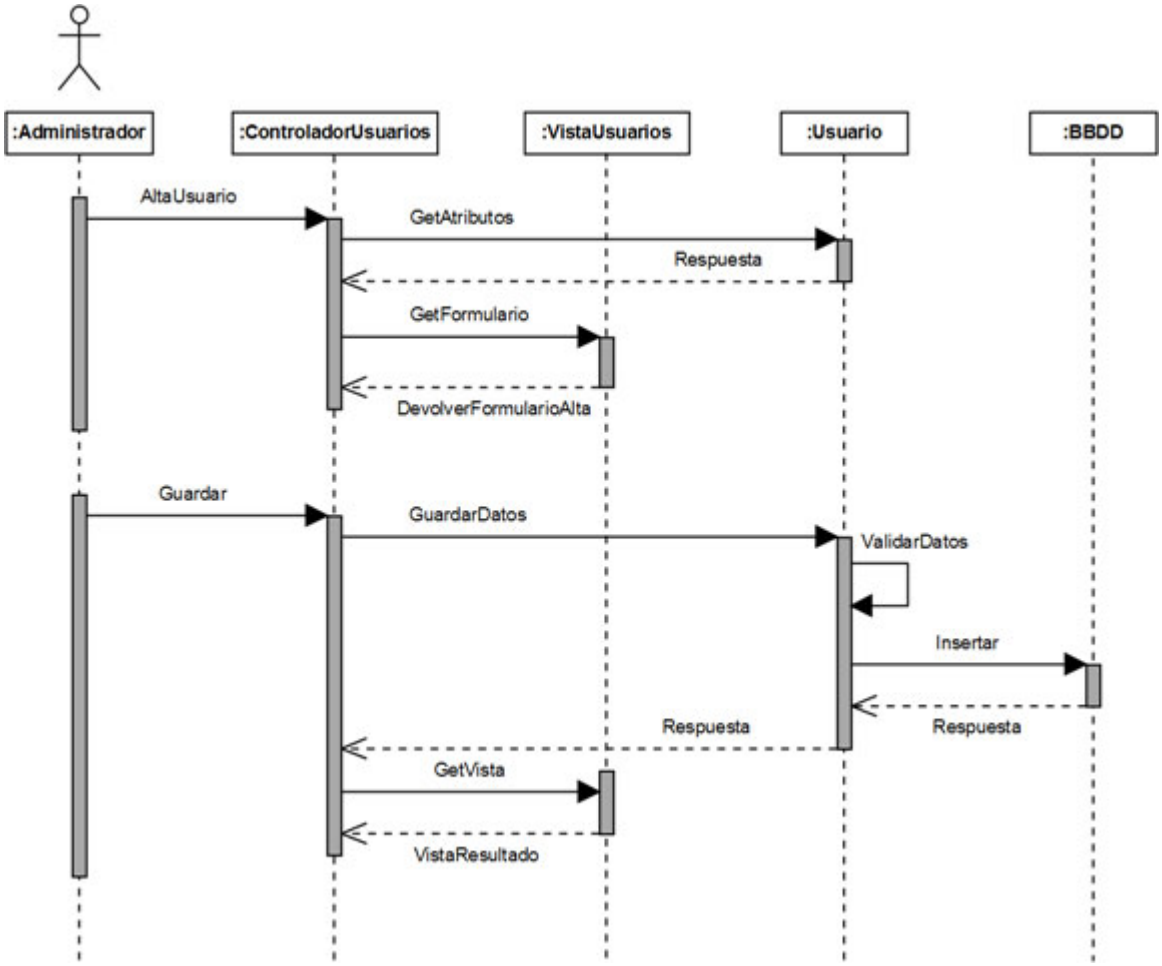
4.5. Diagramas de Secuencia

4.5.1. Diagramas de Secuencia de la aplicación web

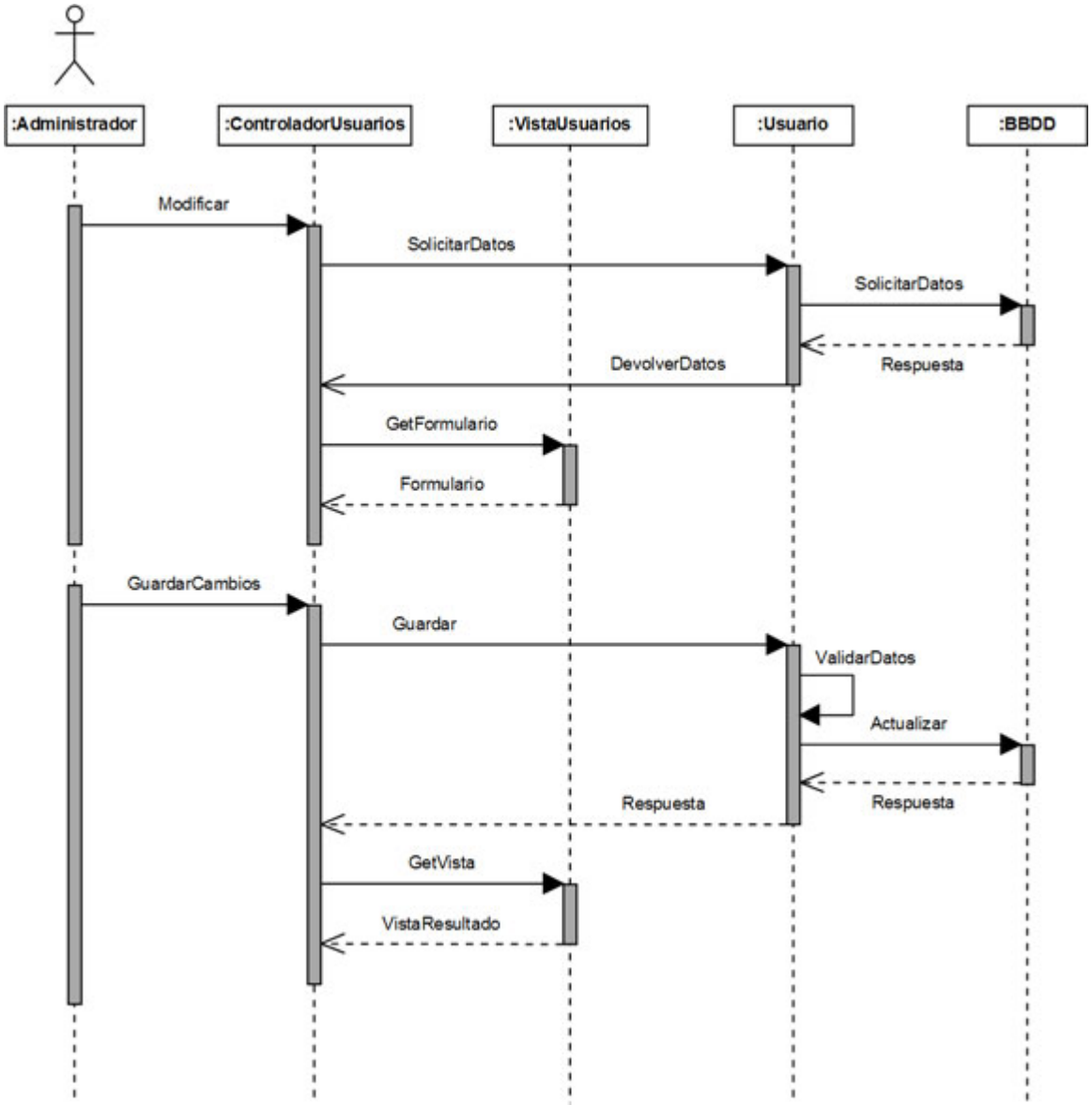
Login



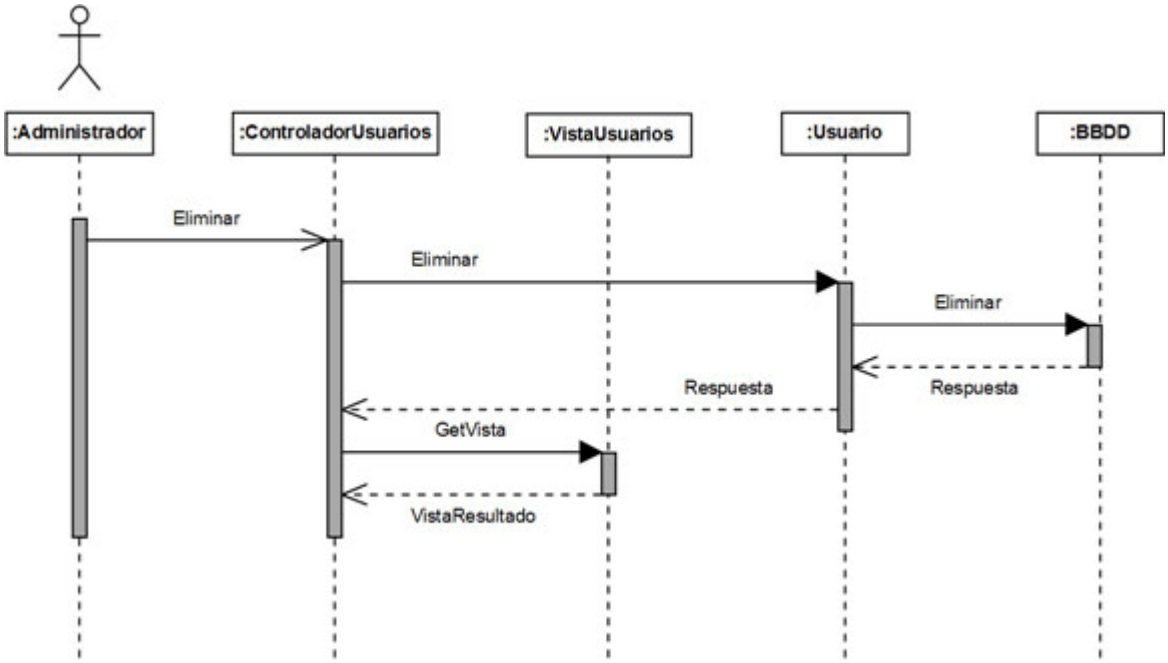
Alta Usuario



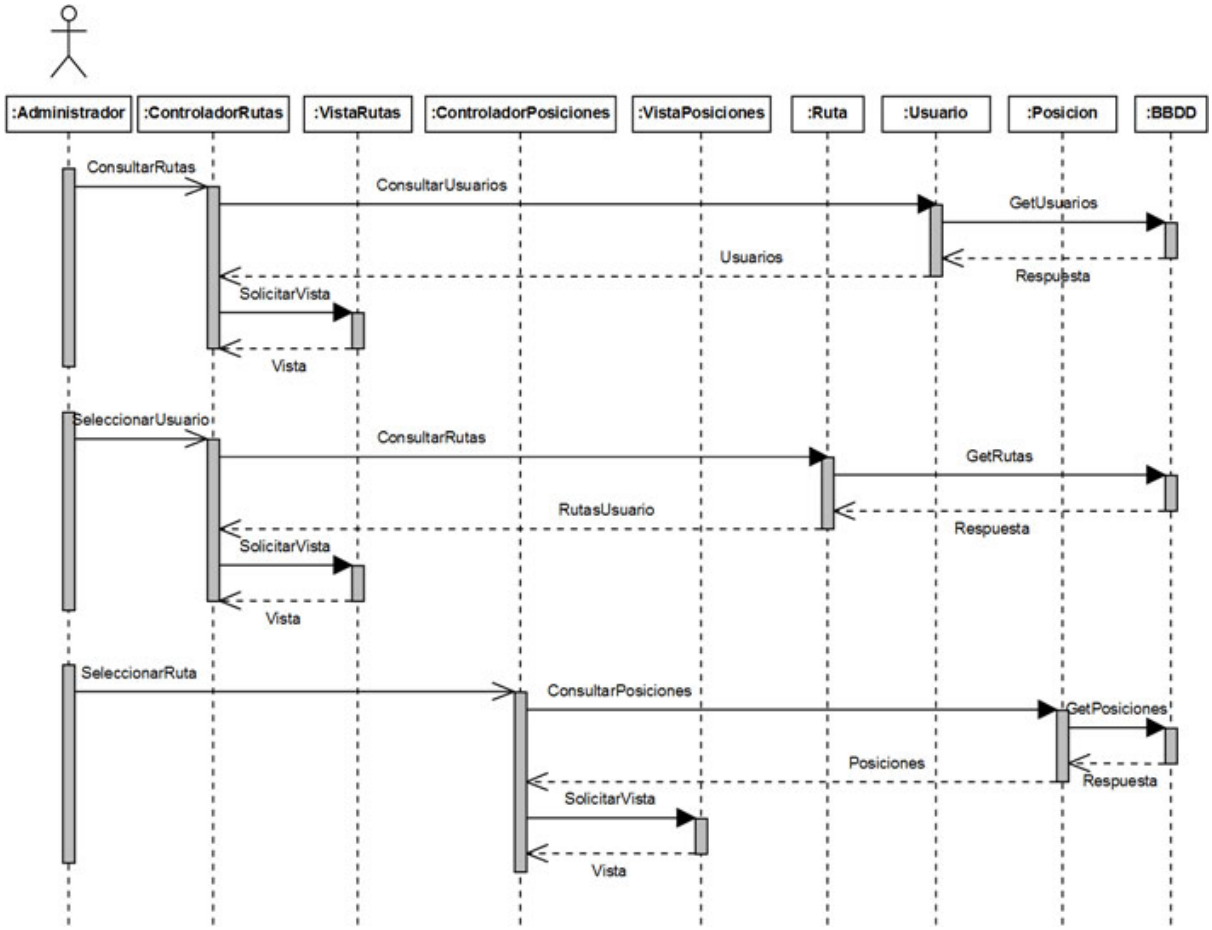
Modificación Usuario



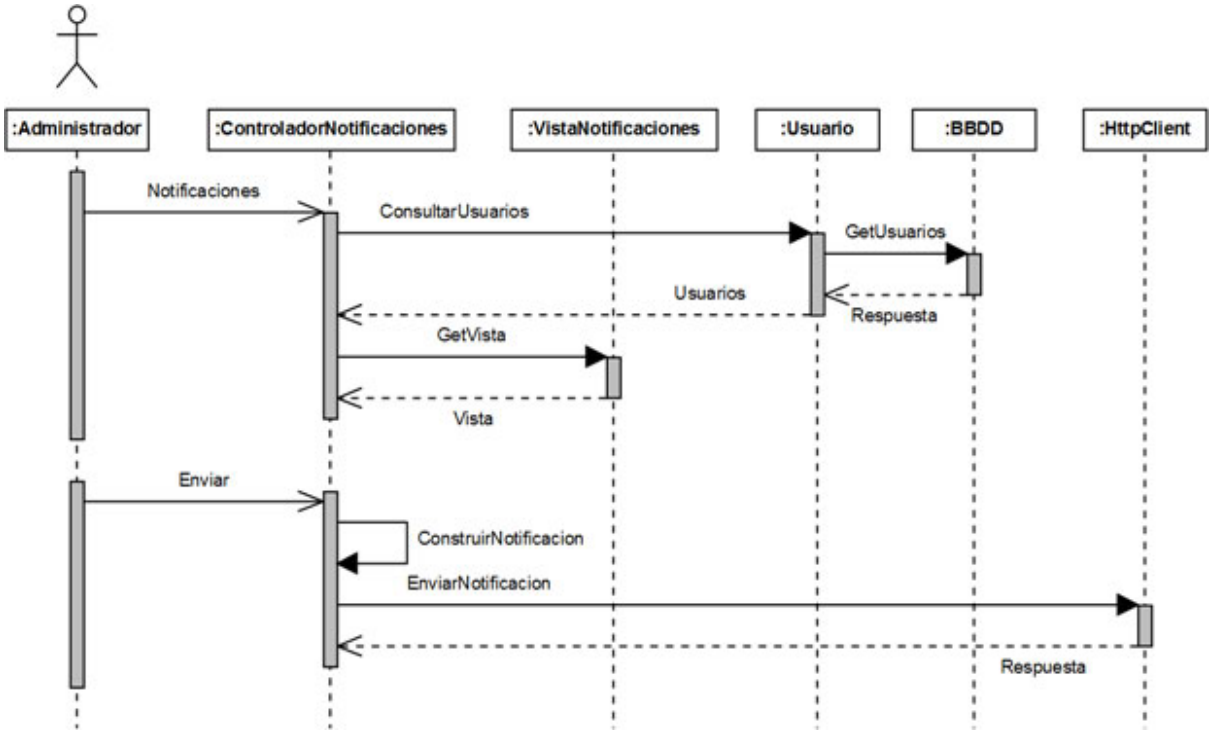
Baja Usuario



Consulta Rutas

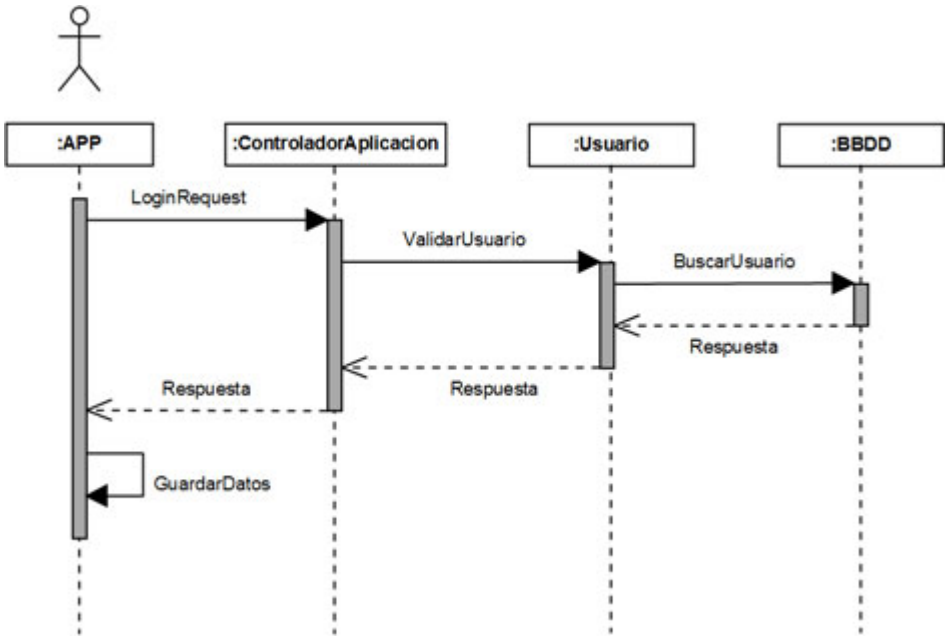


Enviar Notificación

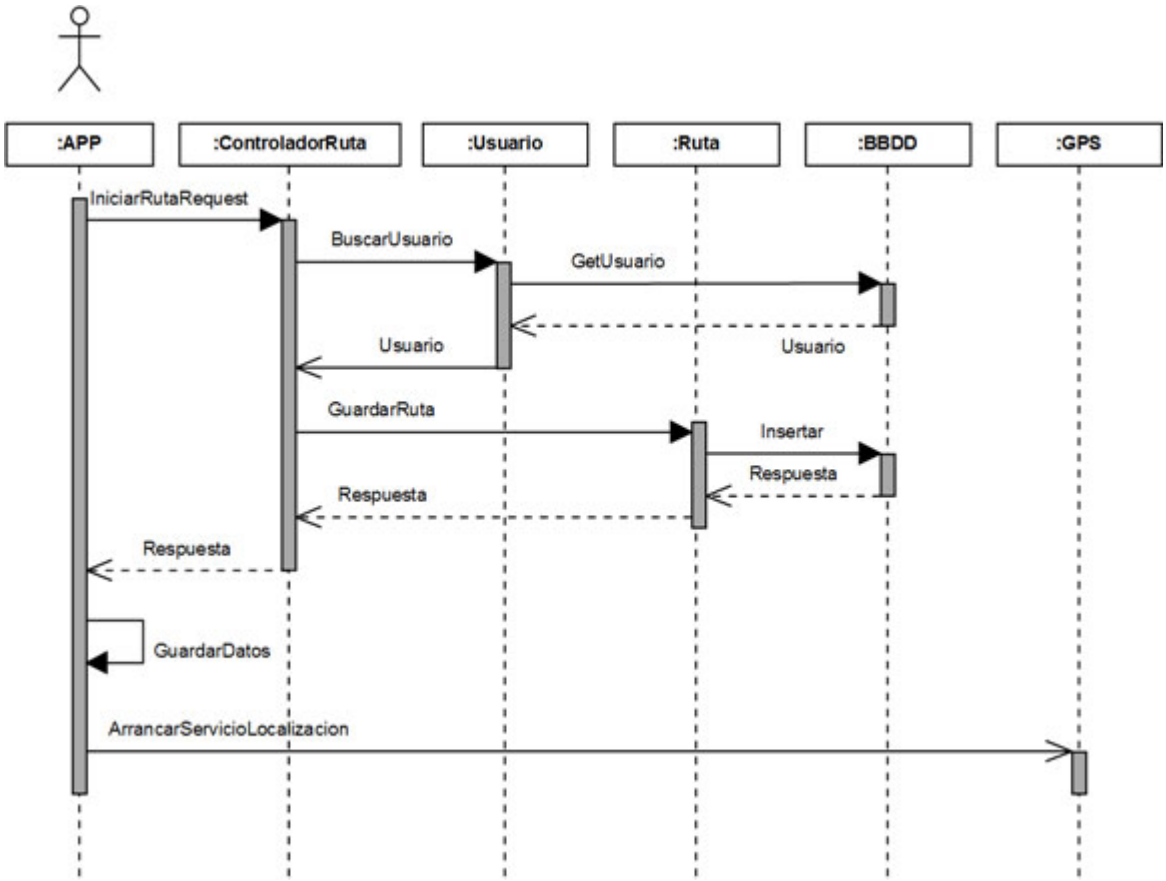


4.5.2. Diagramas de Secuencia de la aplicación Android y servicios web

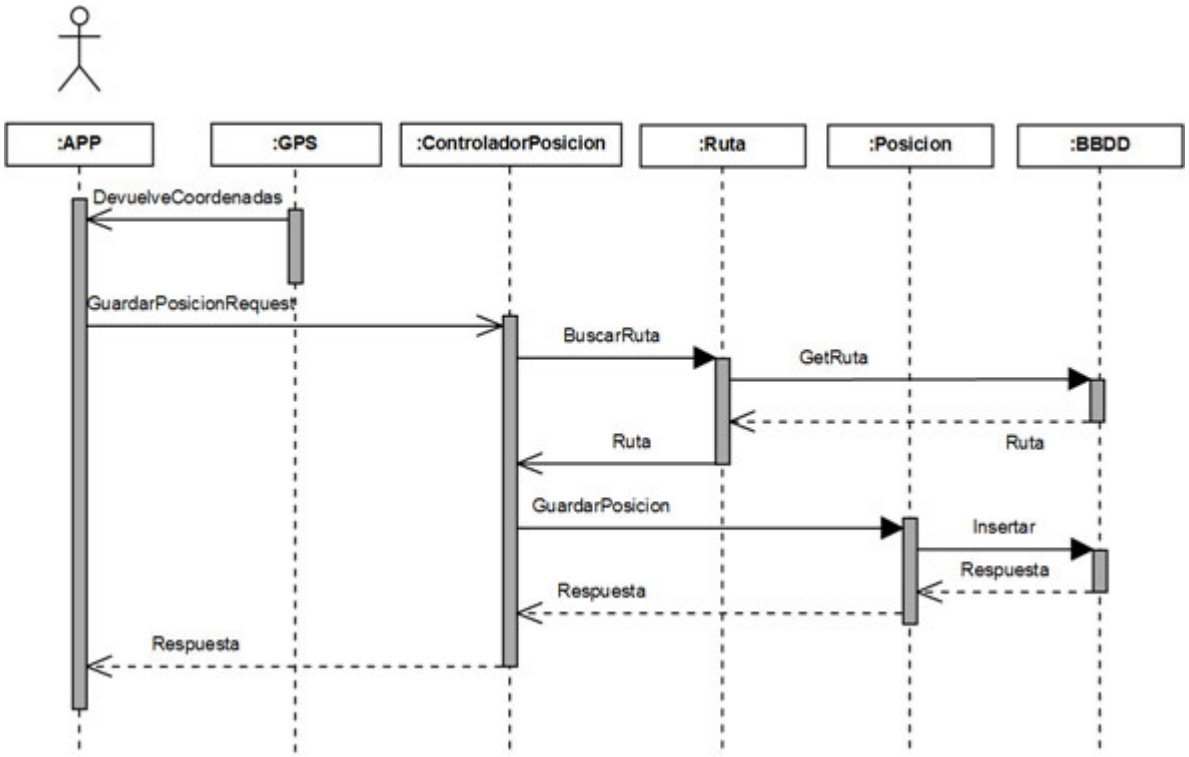
Acceder aplicación



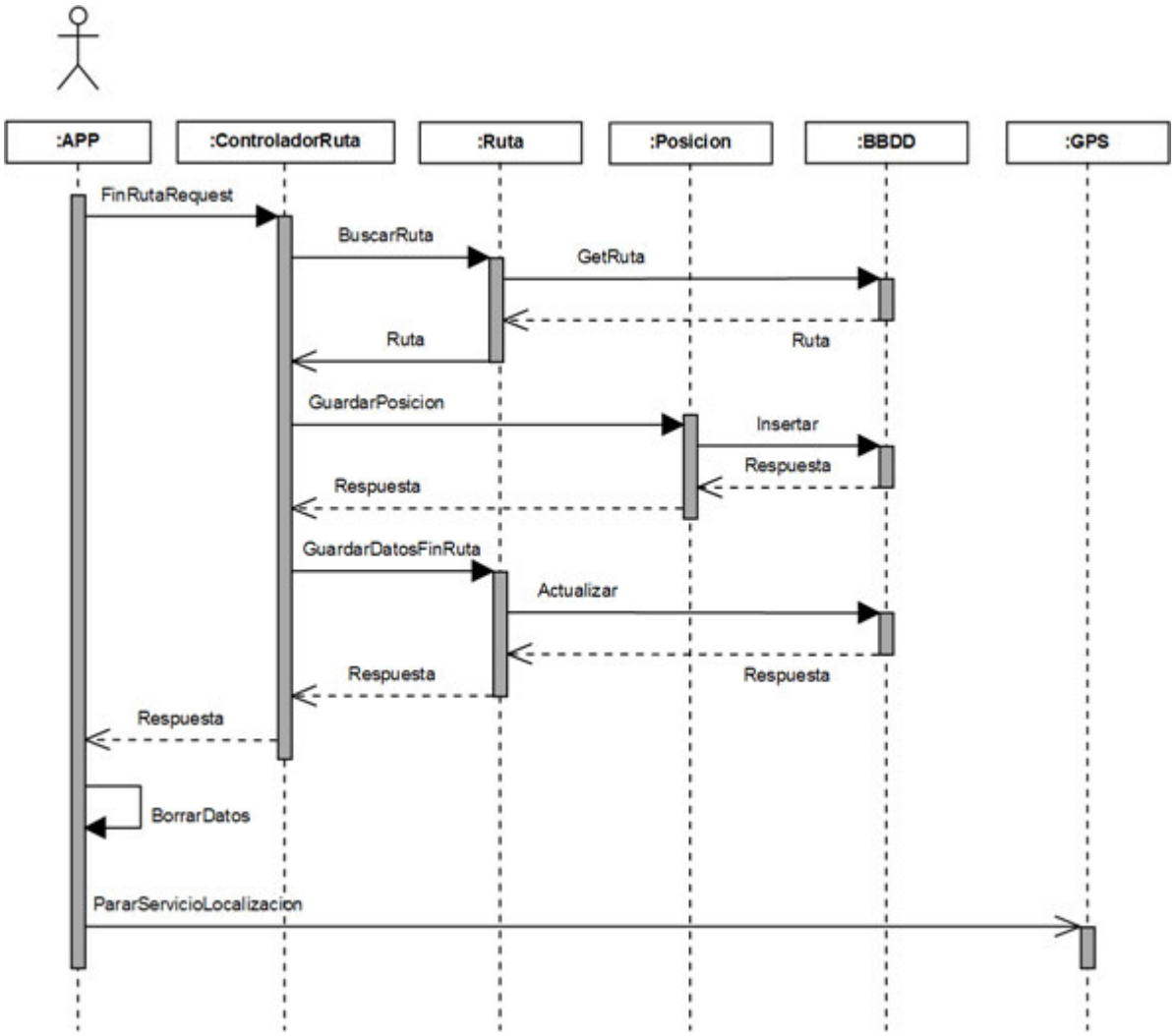
Iniciar ruta



Guardar posición



Finalizar ruta



5. Manual de usuario

A continuación se detallan una serie de instrucciones y explicaciones del funcionamiento y manejo de las dos aplicaciones.

5.1. Aplicación web

La url de acceso a la aplicación web es:
https://obscure-wave-3382.herokuapp.com/

Login

Para comenzar a utilizar la aplicación web, lo primero es iniciar sesión y eso solo lo podrán hacer los usuarios de tipo Administrador.

Si el usuario o la clave son incorrectos se muestra un mensaje informando:

Usuario y/o clave incorrecto/s

Y si el usuario no tiene permisos de administrador también se informa:

No tiene permiso de acceso

Menú principal

Una vez un usuario administrador ha iniciado sesión, se muestran todas las opciones de la aplicación:

- Usuarios del sistema
- Consulta posiciones
- Notificaciones
- Cambiar clave

Usuarios del sistema

Muestra una pantalla con los usuarios del sistema y sus datos.Inicialmente el token device no está almacenado.

Para cada uno de los usuarios se muestran dos opciones:

- Editar

  Para modificar los datos del usuario.

- Eliminar

  Para eliminar a un usuario de la aplicación.

En la parte inferior hay dos botones más:

- Alta

  Muestra un formulario con los atributos de un usuario para completarlos

- Volver

  Botón para regresar a la ventana anterior.

Consultar rutas

Al seleccionar esta opción lo primero que se muestra es una pantalla con un combo para seleccionar un usuario.

Después de seleccionar un usuario, la siguiente ventana es una lista desplegable con las rutas del usuario seleccionado, ordenadas por fecha.

Finalmente, después de seleccionar una ruta, se muestra una pantalla con la ruta dibujada sobre un mapa. En el punto de inicio hay un marcador con el title “Inicio” y si la ruta ha finalizado, se muestra otro marcador con title “Fin”. El mapa se centrará en la mitad de la ruta.

En el lateral derecho del mapa se muestra la fecha de inicio, la fecha de fin y la duración de la ruta. Si la ruta no ha terminado, la fecha fin no se mostrará y la duración de la ruta tampoco se mostrará, en su lugar aparecerá el mensaje:

No ha finalizado la ruta todavía

Notificaciones

En esta ventana se muestra un textarea que permite introducir un mensaje de X caracteres como máximo. Debajo del textarea se muestra una lista desplegable de los usuarios de la aplicación. Se seleccionará uno y se pulsará el botón Enviar para enviar una notificación con el texto introducido al usuario seleccionado.

5.2. Aplicación Android

Login

Lo primero que tendrá que hacer un usuario es acceder a la aplicación con su usuario y contraseña, la cual se habrá dado de alta anteriormente en la aplicación web.

Iniciar ruta

Una vez el usuario se ha accedido correctamente, para comenzar el registro de ubicaciones de la ruta, tendrá que pulsar el botón de Iniciar ruta.

Finalizar ruta

Cuando el usuario haya terminado su ruta, para finalizar los registros de ubicaciones, tendrá que pulsar Finalizar ruta.

6. Conclusiones

6.1. Valoración personal

La realización de este proyecto me ha servido para adquirir una serie de conocimientos que antes no tenía.

Enfrentarme a la programación Android ha sido sin duda lo más novedoso y complicado. Conocer la estructura de una aplicación Android y el uso de su SDK ha sido complejo pero me han ayudado mis conocimientos previos en el lenguaje Java.

Aunque llevo unos años en el mundo laboral, todavía no me había encontrado en la situación de construir una aplicación desde cero, tener que decidir el patrón que seguir para mi aplicación web y buscar un framework que facilitase esa tarea, ha sido una parte que me ha aportado mucho y que sin duda ha añadido valor a mi carrera profesional.

El desarrollo de las dos aplicaciones ha tenido momentos bastante complicados que he tenido que ir solventando a base de documentarme y probar, hasta llegar a la idea que quería desarrollar.

Me ha encantado ver cómo he necesitado y puesto en práctica muchos de los conocimientos adquiridos en la carrera. Y ver como la unión de los mismos da como resultado el proyecto que presento.

6.2. Líneas futuras y mejoras

Una vez que he aprendido a desarrollar aplicaciones Android e integrarlas con servidores web, veo más posibilidades a la aplicación que comencé a desarrollar al inicio.

A continuación detallo algunas futuras mejoras que podrían aplicarse a las dos aplicaciones.

Se podrían enviar notificaciones a varios dispositivos a la vez, modificando la aplicación web para seleccionar varios usuario a la vez y crear un servicio nuevo para enviar la notificación múltiple.

Una mejora sencilla sería poder enviar avisos categorizados desde la aplicación Android al servidor. Se podría usar botones que hagan cada uno una función, una llamada directa a emergencias, un aviso de avería con un mensaje de texto, un aviso de accidente, o un mensaje cualquiera de texto con la posibilidad de adjuntar una foto de un problema. Para esto habría que realizar nuevas peticiones en el servidor y llamarlas desde la aplicación.

También sería sencillo mejorar la consulta de las rutas ya realizadas, por ejemplo por posición inicial y final. Añadir una opción de historificar rutas antiguas o borrarlas. Esto se realizaría en la aplicación web.

Como mejora futura para analizar, se podría añadir una etiqueta NFC en los vehículos para poder asociar al usuario validado en la aplicación con un vehículo y hacer un seguimiento con más información. Al entrar en el vehículo el conductor se validaría en la aplicación y después pasaría el dispositivo por la pegatina, así se recogería la información del código del vehículo almacenada en ésta y se enviaría al servidor. Para esto habría que añadir la lectura de las etiquetas en la aplicación y añadir la información del código del vehículo en el modelo de datos y lógica del servidor, así como en las peticiones que consideremos.

7. Bibliografía

- http://www.android.com/
- https://eclipse.org/
- http://developer.android.com/sdk/installing/studio-build.html
- http://developers.google.com/
- https://github.com/stephanenicolas/robospice
- https://github.com/stephanenicolas/robospice/wiki/Retrofit-module
- http://square.github.io/retrofit/
- http://square.github.io/okhttp/
- http://www.oracle.com/technetwork/es/java/javase/downloads/jdk7-downloads-1880260.html
- http://git-scm.com/download
- https://www.playframework.com/
- https://www.heroku.com/
- http://www.console.developers.google.com/
- https://www.getpostman.com/
- http://www.w3schools.com/
- http://stackoverflow.com/
- http://jsfiddle.net/
- http://www.adictosaltrabajo.com/tutoriales/
- http://www.pacestar.com/uml/
- http://www.yworks.com/en/products/yfiles/yed/
- http://www.littlegreenrobot.co.uk/news/this-sweet-infographic-explores-the-history-of-android-from-cupcake-to-lollipop/
- http://www.worknol.com/history-android-versions/
- http://www.worknol.com/advantages-android-operating
- https://gradle.org/getting-started-android/#av_section_2